Contributions to Big Geospatial Data Rendering and Visualisations

David Anthony Tully

A thesis submitted in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy

April 2017

Declaration

I David Tully, declare that the work within this thesis and research project is original, except where reference is justified to other works and scholars. This thesis has not been submitted for another other degree at Liverpool John Moores University, or any other academic institution.

01.03.2017

[illegible]

Acknowledgments

This work has been produced only by the help and support of friends, colleagues, and family.

I would like to take this time to thank my supervisory team. Professor Abdennour El-Rhalibi, whom without I would not be submitting this thesis. His guidance has helped me through the years of study, not only with my PhD work, but also my undergraduate work. He planted the seed of applying for the PhD scholarship and has set my career path into academia. If it was not for the casual conversation over the shared tangerine, my career and current life would be vastly different.

I must also thank Dr Christopher Carter for whom I have learnt a great deal. His teachings have inspired me to try harder and believe in my own abilities, as well as feeding my thirst for computer games technology. Without him also, my PhD research position would have not been possible; I owe him my career.

I wish to thank Dr Sud Sudirman, a friendly colleague and supervisor, always willing to listen, and work through any problems, no matter how big or small they are, helping me achieve higher standards within my work. His insight into scientific standards and image processing analyse techniques have proven eye opening and productive with the evidence gathering of this work.

I wish to thank all other staff and students at Liverpool John Moores University whom have helped or guide me along the way. A special mention to Dr Madjid Merabti, for whom accepted my application and encouraged me to not be the same as others, but to be myself.

I would like to thank my parents for whom still ask if I am ‘*playing those games*’ at work.

I wish to thank my partner, Nicola Honey, for the understanding and patience during my studies towards the PhD.

To a brighter future, filled with research, and happiness.

List of Acronyms

- ADT – Abstract Data Type
- ALLADIN – Autonomous learning Agents for Decentralised Data and Information
- CGT - Computer Games Technology
- cm – Centimetre
- cm^2 – Centimetre squared
- DSM – Digital Surface Model
- DTM – Digital Terrain Model
- ESRI – Environment Systems Research Institute
- FPS – Frames Per Second
- FXAA – Fast Approximate Anti-Aliasing
- GB - Gigabyte
- GC – Garbage Collector / Garbage Collection
- GIS - Graphical Information Systems
- GPU – Graphics Processing Unit
- IASF - Indexed Array Shader Function
- IVI – Interactive Visualisation Interface
- JRE – Java Runtime Environment
- km – kilometre
- km^2 – kilometre squared
- LiDAR - Light Detection and Ranging
- LoD – Level-of-Detail
- MLAA – Morphological Anti-Aliasing
- OGC – Open Geospatial Consortium
- OS - Ordnance Survey
- OSM – OpenStreetMap
- PBR - Physically-Based-Rendering
- PCG – Procedural Content Generation
- PG - Procedural Generation
- PVS – Project Vision Support
- SAGE – Simple Academic Game Engine
- SAVE – Sustainability Assessment and Enhancement

- SRAA – Subpixel Reconstruction Anti-Aliasing
- SRTM – Shuttle Radar Topography Mission
- TB – Terabyte
- TIGER – Topological Integrated Geographical Encoding and Referencing System
- UI – User-Interface
- WEBVRGIS – Web Virtual Reality Geographical Information System
- WVP – WorldViewProjection matrix
- XML – Extensible Mark-Up Language

Table of Contents

Table of Figures

Table of Tables

Abstract

Current geographical information systems lack features and components which are commonly found within rendering and game engines. When combined with computer game technologies, a modern geographical information system capable of advanced rendering and data visualisations are achievable. We have investigated the combination of big geospatial data, and computer game engines for the creation of a modern geographical information system framework capable of visualising densely populated real-world scenes using advanced rendering algorithms.

The pipeline imports raw geospatial data in the form of Ordnance Survey data which is provided by the UK government, LiDAR data provided by a private company, and the global open mapping project of OpenStreetMap. The data is combined to produce additional terrain data where data is missing from the high resolution data sources of LiDAR by utilising interpolated Ordnance Survey data. Where data is missing from LiDAR, the same interpolation techniques are also utilised. Once a high resolution terrain data set which is complete in regards to coverage, is generated, sub datasets can be extracted from the LiDAR using OSM boundary data as a perimeter. The boundaries of OSM represent buildings or assets. Data can then be extracted such as the heights of buildings. This data can then be used to update the OSM database. Using a novel adjacency matrix extraction technique, 3D model mesh objects can be generated using both LiDAR and OSM information.

The generation of model mesh objects created from OSM data utilises procedural content generation techniques, enabling the generation of GIS based 3D real-world scenes. Although only LiDAR and Ordnance Survey for UK data is available, restricting the generation to the UK borders, using OSM alone, the system is able to procedurally generate any place within the world covered by OSM.

In this research, to manage the large amounts of data, a novel scenegraph structure has been generated to spatially separate OSM data according to OS coordinates, splitting the UK into 1kilometer squared tiles, and categorising OSM assets such as buildings, highways, amenities. Once spatially organised, and categorised as an asset of importance, the novel scenegraph allows for data dispersal through an entire scene in real-time.

The 3D real-world scenes visualised within the runtime simulator can be manipulated in four main aspects;

- Viewing at any angle or location through the use of a 3D and 2D camera system.

- Modifying the effects or effect parameters applied to the 3D model mesh objects to visualise user defined data by use of our novel algorithms and unique lighting data-structure effect file with accompanying material interface.
- Procedurally generating animations which can be applied to the spatial parameters of objects, or the visual properties of objects.
- Applying Indexed Array Shader Function and taking advantage of the novel big geospatial scenegraph structure to exploit better rendering techniques in the context of a modern Geographical Information System, which has not been done, to the best of our knowledge.

Combined with a novel scenegraph structure layout, the user can view and manipulate real-world procedurally generated worlds with additional user generated content in a number of unique and unseen ways within the current geographical information system implementations.

We evaluate multiple functionalities and aspects of the framework. We evaluate the performance of the system, measuring frame rates with multi sized maps by stress testing means, as well as evaluating the benefits of the novel scenegraph structure for categorising, separating, manoeuvring, and data dispersal. Uniform scaling by n^2 of scenegraph nodes which contain no model mesh data, procedurally generated model data, and user generated model data. The experiment compared runtime parameters, and memory consumption.

We have compared the technical features of the framework against that of real-world related commercial projects; Google Maps, OSM2World, OSM-3D, OSM-Buildings, OpenStreetMap, ArcGIS, Sustainability Assessment Visualisation and Enhancement (SAVE), and Autonomous Learning Agents for Decentralised Data and Information (ALLADIN).

We conclude that when compared to related research, the framework produces data-sets relevant for visualising geospatial assets from the combination of real-world data-sets, capable of being used by a multitude of external game engines, applications, and geographical information systems. The ability to manipulate the production of said data-sets at pre-compile time aids processing speeds for runtime simulation. This ability is provided by the pre-processor. The added benefit is to allow users to manipulate the spatial and visual parameters in a number of varying ways with minimal domain knowledge. The features of creating procedural animations attached to each of the spatial parameters and visual shading parameters allow users to view and encode their own representations of scenes which are unavailable within all of the products stated. Each of the alternative projects have similar features, but none which allow full animation ability of all parameters of an asset; spatially or visually, or both. We also evaluated the framework on the implemented features; implementing the needed algorithms and novelties of the framework as problems arose in the development of the framework. Examples of this is the algorithm for combining the multiple terrain data-sets we have (Ordnance Survey terrain data and Light Detection and Ranging Digital Surface Model data and Digital Terrain Model data), and combining them in a justifiable way to produce maps with no missing data values for further analysis and visualisation.

A majority of visualisations are rendered using an Indexed Array Shader Function effect file, structured to create a novel design to encapsulate common rendering effects found in commercial computer games, and apply them to the rendering of real-world assets for a modern geographical information system.

Maps of various size, in both dimensions, polygonal density, asset counts, and memory consumption prove successful in relation to real-time rendering parameters i.e. the visualisation of maps do not create a bottleneck for processing. The visualised scenes allow users to view large dense environments which include terrain models within procedural and user generated buildings, highways, amenities, and boundaries. The use of a novel scenegraph structure allows for the fast iteration and search from user defined dynamic queries. The interaction with the framework is allowed through a novel Interactive Visualisation Interface. Utilising the interface, a user can apply procedurally generated animations to both spatial and visual properties to any node or model mesh within the scene.

We conclude that the framework has been a success. We have completed what we have set out to develop and create, we have combined multiple data-sets to create improved terrain data-sets for further research and development. We have created a framework which combines the real-world data of Ordnance Survey, LiDAR, and OpenStreetMap, and implemented algorithms to create procedural assets of buildings, highways, terrain, amenities, model meshes, and boundaries. for visualisation, with implemented features which allows users to search and manipulate a city’s worth of data on a per-object basis, or user-defined combinations. The successful framework has been built by the cross domain specialism needed for such a project. We have combined the areas of; computer games technology, engine and framework development, procedural generation techniques and algorithms, use of real-world data-sets, geographical information system development, data-parsing, big-data algorithmic reduction techniques, and visualisation using shader techniques.

1 Introduction

In this research we have investigated new techniques for Big Geospatial Data Rendering and Visualisation, leading to the development of new big data processing, correction, interpolation, rendering and visualisation algorithms; and new adaptive spatial data structures to manage efficiently these big-data data-set. In particular, we propose a novel framework called Project Vision Support (PVS) that provides an automated generation and visualisation of real world open data maps for the creation of procedurally generated, highly dense terrain, urban environments, utilising a rendering and game engine.

Real world data-sets are very complex and a pipeline which can orchestrate, combine, and categorise big-data sources of real-world maps into an end product allowing a user to see only what they want to see, in the dimension they want to see it in, is needed for a modern geo-visualisation Graphical Information Systems (GIS). Many GISs are improving due to reduced cost in hardware and graphical processing capabilities, as well as open data provided by many private and public sources. Many GISs are web-based which restrict the real-time rendering capabilities found in modern AAA games which utilise direct access to the Graphics Processing Unit (GPU). They also restrict the interaction with the raw data 'under the hood' of the application, allowing only a window to peek into the data. Other issues are the limitations and restrictions modern GISs apply to the user through user-agreements and limited APIs. Utilising modules found in Computer Games Technology (CGT) such as; 3D and 2D camera systems, multiple projection types, animations, advanced real-time debugging features, custom model meshes, and advanced shader management for forward and deferred rendering techniques, can provide enhanced user interaction and visualisation capabilities. Combined with open data, which has been initially categorised, and the ability to further categorise, adds vast additional user interactions and engagement.

GISs and GIS data have issues such as; complex data sets, limited interactivity with visualisations and simulations provided by the GIS, limited virtual view points within runtime simulations, erroneous data or missing data within data sets, minimal scalability of scenes and extensibility of frameworks. CGT can fix these issues within the GIS domain.

The purpose of this research is to study the development of 3D GIS geo-visualisation and rendering techniques specifically concentrating on combining big-data data-sources of Ordnance Survey (OS), Light Detection and Ranging (LiDAR), and OpenStreetMap (OSM) for the procedural generation (PG) of additional enriched data for the improvement of each original data source, involving error correction using interpolation, and enhancing the realism of visualisations. After combination of big-data for improved data generation, we process and generate procedural geo-referenced objects in the form of; buildings, highways, boundary meshes, waterways, and bodies of water, as polygon meshes. A novel scenegraph has been developed for the organisation of assets within the virtual environment. Scenegraphs are a common structure used within CGT. We researched and developed a custom scenegraph structure for the organisation and categorisation of the different data-sources available. After the Procedural Content Generation (PCG) of geo-referenced assets, we focused on advanced rendering techniques and the utilisation of CGT. Advanced rendering techniques are employed in the form of Array-Index Shader Functions, splitting commonly used shader functions into individual modules. These shader functions are combined in multiple orders to produce realistic, but also interesting and dramatic visualisation results. The shader functions are applied to all geo-referenced objects.

The research culminates in the generation of two main pipelines and additional toolsets. The first pipeline will be responsible to generating the list of geo-referenced objects using the data source e.g. OSM. The second pipeline is responsible for the rendering and visualisation of the geo-referenced objects. Multiple toolsets are generated for the evaluation of novel data-structure and rendering algorithms, and the experimental analysis of the different data-sets. These toolsets are used for intermediate research into the best research and development path to take.

This combination culminates in the improved rendering rates and scene visualisations of real-world data maps into 3D environment using big geo-spatial data.

In this research area, there are similar projects, academic and commercial, which we wish to relate our research too. Extracting beneficial insights from the work already undertaken by the projects of *SAVE* (Sustainability Assessment Visualisation and Enhancement) [1]–[4], *ALLADIN* (Autonomous Learning Agents for Decentralised Data and Information) [5]–[7], *SAGE* (Simple Academic Game Engine) [8], OSMBuildings.Org [9], [10], SimCity Glass Box Engine[1], and the popular commercial computer games *Cities Skylines*[2] and *Planet Rollercoaster*[3]. We aim to improve on particular aspects of these projects, by combining the best of each project and developing novel visualisation and rendering features into a modern advanced GIS.

1.1 Aims and Objectives

The main aim of the project is the processing and visualisation of GIS data to generate a framework capable of parsing, and visualising larges real-world environments. Due to the scale of GISs, we are

[1] https://www.giantbomb.com/glassbox-game-engine/3015-7404/
[2] http://www.citiesskylines.com/
[3] https://www.planetcoaster.com/

investigating how to handle the vast amounts of geo-spatial data and specifically by allowing the processing and visualisation of many data sets at once and in combination, within a single framework; from raw data, to visualisations of complete large urban scenes using advanced rendering techniques. The use of the framework can potentially be used to view any environment within the world if data is available. This data would be obtained from OSM, and OS supported by terrain data from LiDAR data-sets. At a minimum, a single data-set can be used.

The objectives of the research can be summarised by the following tasks:

1. to review and consolidate the knowledge and state of the art on big geospatial data rendering and visualisation
2. to identify and collect the real world data to generate complex real world 3D environment
3. to design framework and software components in order to develop a big geospatial data rendering and visualisation system
4. to create a novel data structure to combine several data sources and design new algorithms to process and visualise these data using a 3D game engine
5. to implement the geospatial data rendering system
6. to develop new metrics and benchmarks to evaluate performance of the system and the realism of the resulting 3D scenes

The overall objective of this research and development project is to develop a novel framework for the production and visualisation of a real-world city's worth of data.

Many real-world products provide the visualisation and rendering of map data, but do not include many features which would prove advantageous. The features we speak of can be found in multiple projects and products, but not within a single self-contained framework.

Having an open and scalable simulation engine, which automatically generate a city's worth of data, can be ported to other languages and other engines with minimal of code changes.

We justify the combination of CGT and GIS for the generation of a modern GIS capable of advanced rendering technique. Game engines provide many of the functionalities, features, and components needed to advance current GIS techniques.

1.1.1 System 1 – Pre-Processor

The first pipeline is responsible for pre-processing the multiple data-sets into an output object which can be used within the runtime simulation we propose. The output will also be able to be used within external software applications such as game engines. The datasets will be parsed into data formats which can be utilised by novel algorithms to extract, organise, and infer data.

Extracting data attributes for the PCG of 3D model mesh objects by algorithms which check the geospatial data, and if issues within the data arise, the algorithm will be able to reconcile the issues to produce data which can be used without additional error checks.

The organisation of the data will be done by creating a novel scenegraph structure which will spatially organise assets within a 1 kilometre squared areas, and through custom layout depicting commonly found attributes within the data structures (buildings, highways, amenities etc.). The scenegraph has another benefit other than spatially organising assets; it allows processing of partitioned areas. These partitions can be processed individually, or used to remove data not needed for processing, increasing processing speeds. Nodes within the scenegraph can be written to data files, and using a middle-out loading mechanism, a large area can be processed, but only a small area needs be loaded at runtime.

Due to the issues surrounding geospatial data, inference of data is needed. If an error exists, or an attribute does not exist, inference from alternative data sources is undertaken; e.g. the height of a building is extracted from the terrain data, whilst the boundary of a building is extracted from OSM. When data cannot be inferred, default attributes will be used.

Parsing the datasets and merging them into a single output format is a key element to this pipeline. This issues the need for a modern GIS framework which can parse, combine, and extract data for use by a user, or by the second pipeline of this project.

With the high-level processes stated, a game engine can provide initial development benefits. These benefits are further apparent within the runtime simulation.

1.1.2 System 2 – Application

The second pipeline is responsible for parsing the output data from the first pipeline into runtime assets, and rendering them in real-time to screen. The application will allow users to create and navigate a virtual environment, and interact with specific objects within the scene. The scenegraph structure used within the pre-processor can be reused within the runtime application, spatially organising assets, and providing iterative methods for searching, categorising and rendering assets over multiple branches.

The application system must improve on the abilities of current GISs such as *Google*, *Bing*, *and ArcGIS* etc. This can be achieved by the combination of game engine technologies;

- 2D and 3D cameras – 2D cameras for overlay systems, 3D cameras for projecting 3D model mesh objects.
- PCG and 3D models loading.
- Advanced lighting calculations and accompanying algorithms – setting the lighting calculations in a processed manner, and dynamically setting this in real-time manners.
- User-interface to interact with the system - the ability to view and interact with GIS data which has currently been negated from many major GISs.
- Input – take input and check for changes of state by the user. We iterate the potential for use of game controllers.

Utilising the components stated, will provide a runtime application capable of utilising the data produced by the pre-processor pipeline will produce a modern GIS framework for an advanced usage.

1.2 Evaluation

The evaluation techniques will consist of iterative processes which takes place over multiple prototypes of the framework as a whole and also the individual modules which make up the framework.

With each iteration of the prototype we will monitor;

- Frames per second
- Update time of each frame
- Rendering time of each frame
- Processing time of specific algorithms – this may consist of large algorithms, as well as the time of the individual methods/functions.
- Rendering time of individual shading techniques

We will screen capture pertinent information due to the visual nature of the project.

To evaluate this against similar projects such as *Google*, *ArcGIS*, *SAVE*, *ALLADIN*, and the work of Bo Mao [11] with his Doctoral thesis work. The evaluation will compare multiple parameters;

- Features – what features are present to the user?
- Components – what components are found within the systems?
- Scalability – can the framework be scaled up easily with varying data-sets?
- Flexibility – can multiple data sets be added easily?
- Realism of visualisations – how realistic does the visualisations appear?
- Data representation – is the data represented accurate and clear?

- Usability – the use of the runtime simulation for multiple hardware configurations.

We will also evaluate the framework on multiple hardware configurations, monitoring the framerate, polygon count, and vertex count. This will determine the scalability with various sized maps, and different hardware configurations.

1.3 Scope and Limitations of Research

We state the scope of the research within this section due to the large scale of the project. Research will be completed within the domain of GIS technologies, computer game technologies, as well as the combination of the two for the development and research of a framework to combine GIS data sets for the advanced rendering and interaction of real-world scenes.

The areas of research within the GIS domain are;

- Research current providers of GIS technologies and data sets. Research the functionalities of the frameworks/services, and research issues or benefits of the geospatial data sets which are open or private.
- Big geospatial challenges with data sets used within GIS technologies and frameworks. The combining and inferring of geospatial data for the creation of complete data sets for use with data processing and visualisation.
- Organisation of data sets for improved storage, and runtime simulation and advanced visualisations.

The areas of research within the CGT domain are;

- Procedural Content Generation techniques for building and scene generation, specifically that of terrain generation utilising real-world geospatial data as a base input.
- Scene organisation to allow real-time rendering of dense scenes.
- Applying CGT to the domain of GIS.

The project will not research the big-data analytics, due to this not being an objective of the current research, but we issue that big-data analytics research is needed as an addition of the project; to be completed within our future work. We justify the reason to not undertake big-data analytics for this project due to the scale of the problem at hand. According to McAfee and Brynjolfsson [12], in 2012, 2.5 Exabyte's of data is produced daily. This figure is estimated to double every 40 months. We estimate that within 2017, 6.25 Exabyte's of data is produced daily. Some of this data consists of GIS data, but also, user generated content such as geotagged images. Deciphering this large amount of data is too large of a task for this project, thusly we iterate that this problem is out of the scope of this project.

With this project, we do not intend to solve all problems within the domain of GIS with the aid of CGT. This is due to the size of the problem perspective, and vast variances of technologies available, as well as large number of various data sources and sets. Our work is limited by the current technologies available during the research stage. The choice of Microsoft XNA as a rendering engine is a limitation due to restrictions for using DirectX9. At the time, alternatives such as HTML5, and UnrealEngine4 were not developed to a satisfactory state for use with this project.

1.4 Contributions

The main novelty of this research is the framework and internal algorithms. The generation of novel data structures and a scenegraph to spatially organise and categorise assets user and procedurally generated from OSM. Algorithms are generated to combine the geospatial data sets to improve each data set and infer additional data to improve categorisations within simulations. The improvement to data, and the inference of data provides additional attributes to categorise procedurally generated 3D model assets. Using the attributes of assets allows advanced rendering of assets depending on the attributes; e.g. rendering buildings which are not fireproof red.

The introduction of Indexed Array Shader Functions (IASF) has not been utilised within geospatial rendering systems or GISs. It is a method used to great success within many AAA commercial games and visualisations applications. The research in combining a static branching shader component with accompanying algorithms for the determination of shader index value for the chosen rendering technique applied to the model mesh component.

Our research determines error reduction methods for LiDAR, and OSM data by combining them independently within iterative processes. The research also determines methods for creating additional data for the LiDAR and OSM data-sets by utilising novel algorithms.

The research has developed a novel storage technique within a formatted texture file by employing parsing procedures to encode and decode the data. This alleviates the use of ASCII files to store the LiDAR and OS data sets, and provides a common format for the viewing of data files.

The use of an Interactive Visualisation Interface allows users of the framework to interact with generated scenes to query combinations of assets properties, categories, and applied user generated procedural animations to both spatial, and visual properties of scene assets. This concept has not been used within GISs to this level of detail; exposing the algorithms and attributes of assets within a scene.

The publications of this research are found in section 1.4.1 and referenced from the numbered list.

Within the journal shown in section 1.4.1, item 1, we present processes for the combination of geographical data-sets. The publication iterates the novelty and acceptance of our proposed framework. The acceptance of the proposed framework is also iterated in the published conference paper in section 1.4.1, item 5.

Within the paper in section 1.4.1, item 2, we introduce our technique for the extraction of model meshes using adjacency matrices by utilising OSM boundary polygons projected from OSM latitude/longitude to 2D X, Y projection, overlaid onto the LiDAR digital terrain maps to copy and extract the data points. These data points are used to create 3D model meshes of individual buildings. A technique which has not been applied using adjacency matrices with LiDAR data and user generated open data.

The utilisation of our framework has been tested by combining real-world noise levels captured within a city within the UK and visualised within the procedurally generated urban environment. The results have been published within in section 1.4.1, item 4. The purpose of the framework is to allow the utilisation of addition overlaid data easily added. This identifies that the framework works with multiple data-set domains.

1.4.1 Publications

1. David Tully, Abdennour El Rhalibi, Christopher Carter, Sud Sudirman "Hybrid 3D Rendering of Large Map Data for Crisis Management", ISPRS International Journal of Geo-Information, ISPRS Int. J. Geo-Inf. 2015, 4, ISSN 2220-9964. http://www.mdpi.com/journal/ijgi/special_issues/disaster-manag
2. David Tully, Abdennour El Rhalibi, Zhigeng Pan, Christopher Carter, Sud Sudirman "Mesh Extraction from a Regular Grid structure using Adjacency Matrix", 8th International Conference on Image and Graphics (organised by Microsoft Research), Aug. 13-16, ICIG 2015, Part III, Tianjin, China. Proceedings Published by Springer Verlag. http://link.springer.com/book/10.1007%2F978-3-319-21969-1
3. David Tully, Abdennour El Rhalibi, Zhigeng Pan, Christopher Carter, Sud Sudirman "Automated Procedural Generation of Urban Environments using Open Data for City Visualisation", 8th International Conference on Image and Graphics (organised by Microsoft Research), Aug. 13-16, ICIG 2015, Part III, Tianjin, China. Proceedings Published by Springer Verlag. http://link.springer.com/chapter/10.1007/978-3-319-21969-1_49
4. William Hurst, Graham Davis, Abdennour El Rhalibi, David Tully, Zhigeng Pan "Predicting and Visualising City Noise Levels to Support Tinnitus Sufferers", 8th International Conference on Image and Graphics (organised by Microsoft Research), Aug. 13-16, ICIG 2015, Part III,

Tianjin, China. Proceedings Published by Springer Verlag. http://link.springer.com/chapter/10.1007/978-3-319-21969-1_53

5. David Tully, Abdennour El Rhalibi, Madjid Merabti, Yuanyuan Shen, Christopher Carter "Game Based Decision Support System and Visualisation for Crisis Management and Response", 15th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, June. 23-24 2014, Liverpool, England. Proceedings Published by Liverpool John Moores University. http://www.cms.livjm.ac.uk/PGNet2014/papers/1569961593.pdf
6. David Tully, Abdennour El Rhalibi, Christopher Carter, Sud Sudirman, "Generating a Novel Scenegraph for a Modern GIS Rendering Framework", IEEE DESE 2016 - Developments in eSystems Engineering, 31st August – 2nd September2016 Liverpool and Leeds England.

1.5 Thesis Structure

The thesis is laid out as such;

Firstly, in Chapter 0, we will discuss the background research needed to understand and become familiar with the cross domain knowledge needed for the project. We discuss the areas of; real-world map data-sets, modern GISs', similar projects to this work as well as commercial products, CGT with visualisation and PG techniques as well as data structures commonly used within computer games, but not within GISs'.

We then present, in Chapter 3, the completed literature review which compares, and analyses modern GISs and their platforms, features, and products. We also critique the work of similar research projects and commercial products, as well as data-sets of OS, LiDAR, Photogrammetry, and OSM for the use within a modern GIS framework. The analysis of games technologies and closely related algorithms/data-structures e.g. visualisation and spatial organisation techniques are justified for a modern GIS framework.

Due to the limitation and issues surround big-geospatial data, we discuss these issues within its own chapter, in Chapter 4. Within this chapter to critique the need of data-sets over similar but consequently vastly different data sets. We discuss the OS data-sets provided freely by the UK Government, and the chosen data-set which is needed for the proposed framework. We also present the details and issues surrounding the 6.7 million pounds' worth of LiDAR data we have obtained from the UK Government. This is followed by an in-depth look at the OSM data-sets and internal data-structures which are used for much of the PG of 3D assets. The restrictions and issues of 3D model mesh assets are presented next.

After the background research, the critical analysis and discussion of others work and related research, as well as the data-sets to and issues of geospatial data we discuss the specifications needed to overcome these issues, Chapter 5. The specification will be discussed from the perspective of the framework which will be developed, and the needs to overcome the problems and issues.

Using the specification, the design section can be completed.

The design section, presented in Chapter 6, is used to showcase the novel techniques and data structures which we have created to complete the project and the design of two pipelines which build up the framework. We introduce the designs for the novel scenegraph data structure, and the novel IASF data structure. When combined these two data structures allow the user to modify a scene to change the look and feel of the scene as a whole, or only subsets of the scene, by the dispersal of a single numerical value. Currently this technique has not been used within a GIS.

After designing the individual components and algorithms needed for the framework, Chapter 7 discusses the implementation of the pre-processor algorithms and APIs used within the framework to parse, combine, infer, and generated big geospatial data.

Chapter 8 presents the implementation of the runtime algorithms used for searching, and rendering the visualisations. Within this chapter we also cover the development of search techniques for the custom objects we have created by parsing the data-sets we have. Combined with the searching of assets within the simulation, the use of a Dynamic Expression Library is used. This allows users to search for objects within a scene using any combination of properties or attributes attached to an asset; given great flexibility for search technique on a large scene. We state how the separate libraries of; input, camera systems, interpolation library, triangulation algorithms, dynamic expression library, pre-processor and runtime simulation, an Interactive Visualisation Interface (IVI), novel scenegraph and IASF interact with each-other. This is also covered within the design section, but the implementations picture the intricate cohesion needed between systems.

Due to the large domain spread of the project, we state the issues and problems which arose during the implementation phase, and specify the issues which need to be researched further.

To conclude, if project and framework is to be successful, the evaluation chapter is presented in Chapter9. Within this section we evaluate the framework against that of professional and real-world GISs as well as similar research projects. We justify the framework by comparing the features of our framework against the features provided by the alternative GISs, frameworks, projects, and pieces of software. We also evaluate the stress testing of the framework by uniform scaling analyse. The frame rate parameters are also captured for 5 maps which are also visually compared with the alternative GISs.

Once the evaluation of the framework is completed, we reach a conclusion in Chapter 10. Chapter 10 includes the future work and directions of our research.

The reader can also find the publications of this project within section 1.4.1. Each publication will be referenced as item 1 to 6.

2 Background

Within this chapter, we discuss our background research and related topics. We concentrate our research on the state of the art modern GISs and the relevant technologies, techniques, and data-sets included within them. We discuss the uses of PCG, and rendering capabilities of similar projects as well as data organisation techniques, how other researches use data commonly found within GISs, and how they combine and exploit this data with CGT and rendering algorithms.

This chapter contains a discussion about visualisation techniques used within similar areas of research, and a short discussion on modern GISs and similar projects as well as data used within GISs. We also review commercial projects which we will compare to our project framework highlighting similarities and improvement, which will bring us upon computer game technologies and what they provide for a project of this type. Within computer games technologies, particular rendering structures are used with varying effect, some needed, while others are not needed for the creation of a modern GIS. Data is needed to be spatially organised which brings the chapter to a close with a discussion of scenegraph structures.

2.1 Map Data

Within this section we will discuss the possible data sets we utilise within this research. The data sets will be discussed as to where the data is being used within the research, the issues (if any) surrounding the use of the data, and the need to incorporate the data into a modern GIS. We start by discussing the OSM mapping platform. After this, we discuss terrain data in the form of LiDAR which can be used in a multitude of ways, primarily the generation of terrain model meshes for visualisation. LiDAR data is also used for information extraction and analysis to combine with other data sets such as the UKs OS data set. This data can provide a base data-set to use when errors are found within the LiDAR data set.

We must evaluate, and be aware of the implications of using map data, be it, volunteered, or open, due to the possibility of errors within the captured data. Droj, Suba, and Buba in [13] state in their opening statement that for centuries of map data capturing, the data is subject to random, systematic, and gross errors. They include their main concerns with using GIS data, and its spatial data quality:

1. There is an increasing availability, exchange and use of spatial data.
2. There is a growing group of users less aware of spatial data quality.
3. GIS enables the use of spatial data in all sorts of applications, regardless of the data quality.
4. Current GIS offers hardly any tools for handling spatial quality.

5. There is an increasing distance between those who use the spatial data (the end users) and those who are creating the spatial databases.

They also presume that the use of GISs will prevail even if the data used if erroneous, leading to misled decisions, and error prone results of the processed artefact. Although we agree with this statement, we do believe that efforts are being made to reduce errors within data capture and processing. The OSM platform, for example, has many tools and standards for checking data within their data-bases. These include the following systems:

- MapDust[4]; a bug tool operated by Skobbler GmbH (a German company), is a bug reporter tool which allows users to submit problems with the OSM navigation results. This itself is allowing users to submit changes, but they are checked manually.
- JOSM Validator[5]; a data validation system which checks many subsets of data. It checks for; duplicated nodes, duplicated nodes within ways, unconnected coastlines, incomplete ways, nodes with the same name, overlapping areas, and many more. We will need to implement similar techniques, or implemented a data processing technique which includes parsing the data through JOSM or a similar format.

Pringle [14] states that processing of spatial data can incur errors. The author suggests specific situations from converting from raster files to vector files; two commonly used data structures for storing spatial data pertinent to GIS. Converting one data structure to another needs specifically designed, and tested parsing algorithms. Even with testing, conversion of one data type to another still has erroneous issues which can distort the validity of said data set. When parsing open data within the framework, vigorous iterative testing must be undertaken to prevent future errors.

With the intentions to test the presumable need for parsing algorithms, to convert open-data and spatial data to runtime class object for the framework, we have created novel techniques to reduce errors within the data-sets use.

2.1.1 Open Street Map

OSM is a worldwide open data mapping project using over 1.7 million volunteers to map local environments. This includes Topological Integrated Geographic Encoding and Referencing system (TIGER) which includes road data for almost all of the USA, which has been donated to OSM.

Within a comparative study done by Haklay [15] from the Department of Civil, Environmental, and Geometric Engineering, University College London, comparing OSM data-sets and OS data-sets, the author states that OSM information is accurate to within average of 6m to that of OS data. Within

[4] http://www.mapdust.com/

[5] http://www.giscorps.org/documents/UsingJOSM.pdf

the research they state that 29% area coverage of England has been captured. This result was from 2008 to 2012. We estimate this number to have increased with the popularity of current OSM affairs and what are known as 'OSM mapping parties[6]'. The author reiterates the needs of volunteered geographical data from Oort's Doctoral Thesis [4]. These qualities needed by open spatial data, specifically volunteered are:

- Lineage/historical: accuracy of data and data capture techniques.
- Positional accuracy
- Attribute accuracy
- Logical consistency
- Completeness
- Semantic accuracy
- Usage, purpose, and constraints
- Temporal quality

These are further iterated by Droj, Suba, and Buba in [13].

OSM data has been used in several GIS projects and used in the early research of Togelius with his Open Games [16][17][18][19]. In [16], the author discusses games which have Open Data as the base of the projects. Interesting uses of this data have been made to generate real-world simplistic city's and highway structures for use in a racing game, and generating interesting real-world Monopoly[7] board game layouts, as well as early Civilisation[8] style computer games which use the boundary data from OSM to generate the terrain map layouts for the game. The racing game which utilises procedural techniques is of particular interest. This work shows that OSM data can be used for the creation of simplified urban scenes, which resemble real-world locations.

OSM has four main elements:

- ***OSMNode*** (a latitude and longitude location on the Earth's surface);
- ***OSMWay*** (a group of ordered *Nodes*. If the first OSMNode is the same as the last OSMNode in the ordered list then this denotes a *boundary*, else it is a *Highway*. The direction of the *Highway* is the order of the *Nodes* in the list);
- ***OSMRelations*** (a list of nodes or ways, but not both, which explain some kind of connection such as a group of buildings, or outlines of buildings which are contained within one another);

[6] http://wiki.openstreetmap.org/wiki/Mapping_parties
[7] https://en.wikipedia.org/wiki/Monopoly_(game)
[8] http://www.civilization.com/en/home/

- ***OSMTag*** (extra information describing the OSMNode or OSMWay, for example *Tag=Building*) which is a key/value pair.

Using only these 4 data-types, a world full of mapping data can be generated. Parsing this data for a custom system used to generate 3D structures of 2D latitude and longitude locations with descriptions is a difficult process; a process which is needed for a novel modern GIS framework.

There are currently a few projects which are tackling the procedural uses with generating more accurate realistic environment.

OSM-3D.org[9] combines OSM with elevation data captured by the Shuttle Radar Topography Mission (SRTM) for the development of a 3D Geodata Infrastructure (GDI-3D). They intend the project to be a world-wide mapping scheme utilising open web service standards of the Open Geospatial Consortium (OGC). Although web technologies are progressing and becoming capable of reaching real-time rendering speeds which closely matches that of desktop GPUs, they currently lack the hardware capabilities. A modern GPU can vastly outperform web technologies for real-time rendering. We plan to take procedures and experience from the OSM-3D.org project, and combine them in a desktop application with use of high a performance GPU and game engine specifically designed for real-time rendering.

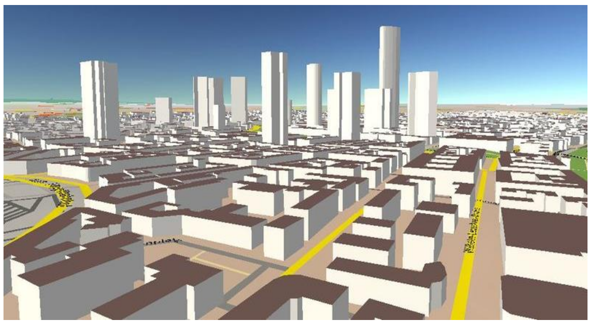


Figure 1 OSM-3D.org project. Image shows 3D procedural buildings with height captured from the Shuttle Radar Topography Mission data. Image obtained from http://wiki.openstreetmap.org/w/index.php?title=OSM-3D.org&oldid=1395279

OSM-3D.org is also working towards generating 3D building models from the OSM data. They utilise simple primitives extruded from the data points given. If a building outline satisfies simplistic rules, a premade rooftop is scaled, orientated and placed on top of the building. This is a basic pitched roof structure. Other projects are working specifically on the procedurally generation of rooftop structures, as well as the mapping techniques needed.

[9] http://wiki.openstreetmap.org/wiki/OSM-3D.org

Although OSM-3D utilises SRTM, the terrain is untouched and only the heights of the buildings are used as shown in Figure 1. Integrating accurate terrain models will improve the visual accuracy of these scenes.

Although to create further realism within a scene, accurate building models are needed. If a user's camera perspective is above a city, as most GIS allow a user to do, the roof tops play a major part for the realism of a scene. For this research we will not focus on the generation of techniques to develop procedural techniques for building tops, but do insist that this is an area which needs completing to generate a complete modern GIS.

We focus primarily on the generation and visualisation of building models. Using the work from OSM-3D.org, we have a good starting point into what OSM can be used for with regards to 3D building model meshes.

To generate building model meshes, and other model meshes from the open geo-data of OSM, processes are needed to arrange and gather specific information from the database. This is a large task due to the large databases of OSM which classify it as Big-Data.

Currently OSM map data contains over: 52,971 different *key* items[10]. A key is a description which can be entered by a user. A key is user generated and a large issue with OSM due to the fact that a key can represent anything, and multiple keys can correspond to the same instance due to multiple languages, local dialects, and spelling mistakes or variant grammar layouts.

The complexity of deciphering the massive data sets and intricate naming's for seemingly similar objects in the world is demanding and needs vast attention to detail to compare names and split corresponding data sets into more manageable groups of data. For example, the *Key: Building* has 7512 *Tags* to describe it, many of which are repeated but with slightly different spellings. Many fuzzy text matching algorithms have been created to fix this problem [20]. Another option would be the manual selection of the most commonly used *Key* and *Tag* data.

Within the OSM database there are:

- 2.8 billion objects
- 1.1 billion tags
- 2.5 billion nodes of which 3.73% have at least one *Tag* description
- 255 million *Ways*, half of which are closed (boundaries)
- 2.8 million *Relations*

[10] https://taginfo.openstreetmap.org/keys

At the time of writing, the global data storage of OSM data is 498Gb. Pre-processing and separating this big-data set can greatly reduce storage space needed for a real-time application. This can be done by utilising scenegraphs commonly use computer games.

As shown in Table 1 and Figure 2, the OSMNode density of the chosen locations varies and gives surprising results. Table 1 depicts 2 city locations; *Liverpool UK* for its familiarity, and New York City – New York due to the common global thought that it is a large and dense city. Within the Table 1 is also a selection of country side from the UK.

The OSMNode count is the number of Nodes within a file. The OSMWay count is the number of Ways within a file. The building count is the number of buildings within a map.

We can notice the difference between the cities of Liverpool and New York 1km^2 maps file size. This shows that even though an area may small in terms of coverage, the density of data within these areas can be large. Utilising spatial partition trees, and parsing techniques to remove data which is not pertinent to the generation of a modern GIS is needed. We bring the reader's attention to the OSMNode density within the various maps of Liverpool UK, Manhattan New York, and the selection of UK country-side at the location of Latitude -1.8697 to -1.8461, and Longitude 53.5048, 53.4900, an area right of Manchester UK. The OSMNode density of the selected country-side file is large compared to an OSMNode density rich area of Manhattan New York. The reason for this is selected maps of OSM include all nodes of a boundary, and if a selected area contains a boundary which is not completely contained within the selected area, all nodes of that boundary are downloaded. For example, downloading a map on the coastal line of the UK will include the nodes which depict the coastline of the complete UK.

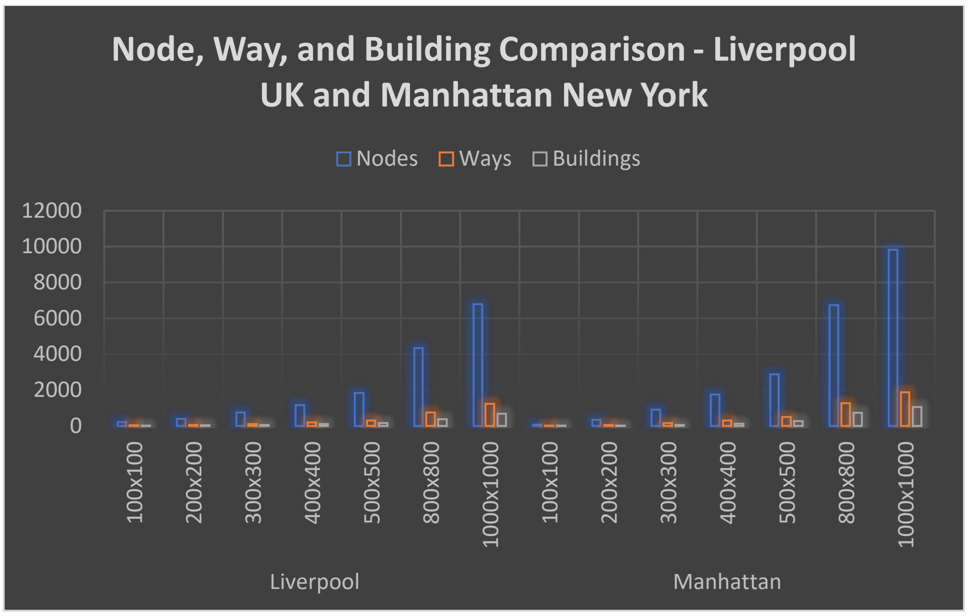


Figure 2 Bar Chart comparing Liverpool UK and New York USA areas depicting OSMNode density, OSMWay density, and Building density.

Name	Area Coverage	File Size on Disk	Node Count	Way Count	Building Count
Liverpool	100x100	64.1 kb	209	24	15
	200x200	108 kb	383	53	33
	300x300	198 kb	747	105	57
	400x400	304 kb	1152	197	103
	500x500	474 kb	1842	300	157
	800x800	1.09 Mb	4338	750	367
	1000x1000	1.74 Mb	6794	1236	679
Manhattan	100x100	24 kb	66	7	2
	200x200	159 kb	333	58	19
	300x300	323 kb	903	149	51
	400x400	579 kb	1756	307	128
	500x500	898 kb	2881	490	261
	800x800	2.63 Mb	6740	1259	722
	1000x1000	3.61 Mb	9827	1875	1043
UK Country-side	1000x1000	843kb	3824	106	0

Table 1 OSMNode density maps within Manhattan New York American, Liverpool UK, and randomly selected country-side within the UK.

An issue with the data is lack of information attached to a building boundary. For this reason, utilising OSM for the PG of accurate city model meshes for a modern GIS is currently not the best choice. Creating a realistic and accurate city skyline where the buildings are of accurate height is the pinnacle of what is needed for a modern GIS. Currently OSM is unable to provide a city's worth of height data.

Combining OSM with terrain mapping data, specifically that of photogrammetry or LiDAR can produce this eagerly needed data. The next section covers LiDAR. We will discuss in a later section the issues surrounding photogrammetry, and why we wish to not pursue it for this research.

2.1.2 Light Detection and Ranging – LiDAR

LiDAR is the capturing of height data from laser scanning cameras commonly aboard low flying aircrafts (800meters); some as small as commercial drones. Pulses of light are shot from the camera, and the time taken for the light photons to travel back to the cameras light sensitive sensor is recorded. Using this time fragment with the addition of gyroscope technology, and GIS coordinates, highly accurate data can be captured; to within a few centimetres of accuracy with 0 – 15cms of error.

There are two types of LiDAR map, Digital Surface Model (DSM), and Digital Terrain Model (DTM). A DSM contains all points captured but with minimal processing from the provider to correlate raw data-points. The DTM holds a processed version of the DSM which contains only height data for terrain with removed buildings, trees, or other artefacts within a scene. The LiDAR data maps are 2-dimensional grids of height points, thusly only the surface of a landscape, from a top down perspective can be captured using this type of LiDAR capturing. This implements errors due to not being able to record inside openings such as caves, or the underside of bridges. When visualised, the high density data maps of LiDAR can produce realistic top-down representations of city models when generated as a 3D model meshes. Combining these city models with a 3D camera system, users can navigate areas of these 3D visualised 2D grids of height points. What cannot be done from visualised LiDAR alone is to depict object information such as, buildings, trees, vehicles etc. This needs further processing and analysis techniques. Combining with other descriptive data sets such as OSM, chunks of data can be extracted.

LiDAR has other issues due to the cost of capturing the data. Although, small aerial drones with LiDAR equipment attached is considerably bringing down costs. Capturing data at multiple resolutions varies the cost of data fragments, thusly many providers provide low resolution LiDAR maps. In this instance, low resolution data capturing stipulates that the distance between the data points are 2 meters squared. This is still highly accurate compared to freely available DTMs provided by the UK government in the form of Open Survey (OS). See the next section on OS.

Many of the areas of the UK have been captured with LiDAR; see Figure 3. As stated, the varying cost of areas has led to the situation of various areas being mapped at different resolutions; high resolution capture for dense urban environments, low resolution for forest areas and none-built areas. These resolutions vary on a company/provider basis. This is an issue for a modern GIS. The UK needs full coverage at high resolutions. If this is not possible then procedures must be put in place to produce accurate data from combining with other sources such as OS or by interpolation techniques to produce additional height data points for higher resolution maps.

LiDAR also has another issue of being incomplete, error prone, and even missing large data-sets. This is for a multitude of reasons. The beams of light do not reflect off organic matter such as vegetation, nor does it reflect accurately off water; producing errors or a complete lack of data. If small parts of LiDAR data are missing, interpolation techniques can be used to generate missing data. If a larger part of a map is missing, then procedures are needed to combine other data-sets for the combination and generation of additional data. OS data can provide a base data-set for the combination. OSM can provide additional data to improve area classification. Area classification can be used to lower height points of LiDAR or OS if it’s determined that the area is terrain or water; producing further accurate data and reducing/removing errors. Capturing data above urban environments is thwarted with further issues. Cities are not still at capturing time; they are a constantly moving entity. This movement is an issue, especially if a flock of birds are flying at time of capture. The beams of light bounce of the birds inflight and data depicts the heights of the birds. When visualised, this introduces huge spikes into the terrain data. Firstly, this is not accurate, but technically the LiDAR capturing equipment is not to fault. Procedures can be put in place to detect this error and flag the situation for manual editing. To create a procedural technique, OSM data can be queried, and if this spike in height data is within bounds of some sort, e.g. a building boundary, then the data can be classified as accurate, else, either we can remove the data within the OSM boundary from the LiDAR map, or pass the issue to a manual operator.

Within the next section we introduce the UK’s OS data, specifically the height data for which they provide freely because there are multiple forms of data and provider.

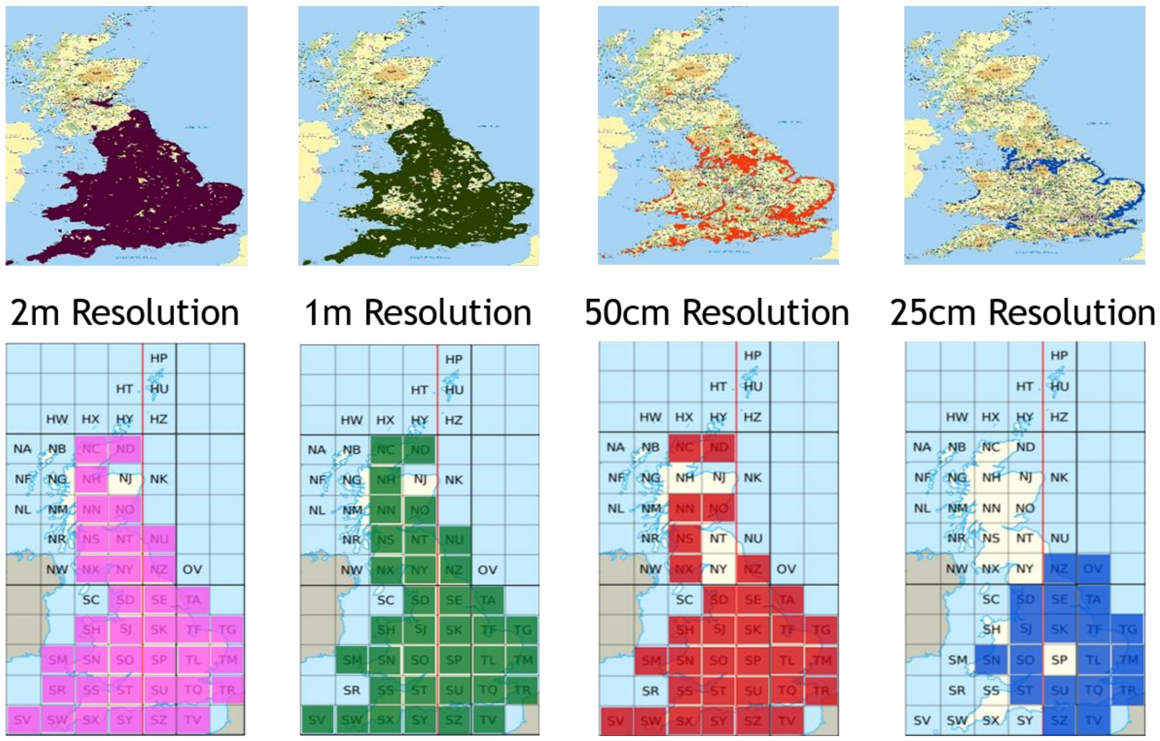


Figure 3 Area coverage of LiDAR data at 2m (Purple), 1m (Green)), 50cm (Red), and 25cm (Blue) resolution. Image obtained from now disbanded government sector.

2.1.3 Ordnance Survey

There are many providers of OS data, but we concentrate on the freely available UK government generated DTM. An official UK government generated guide is available [21]. We discuss shortly the other providers and the issues with the data they provide as well as the reason we have selected the freely available version provided by the UK government. We also discuss the issues surrounding OS, and what can be done to fix these issues.

OS is a mapping scheme provided and updated by the UK government. Its resolution is low compared to that of the LiDAR data; 50ms between each height point. This data is of too low a resolution to use with a modern GIS due to lack of realism generated maps brings to a project. We plan to use interpolated OS maps in place of missing LiDAR maps. OS can also be used to fill in missing data points of the LiDAR which do not satisfy certain classifications; i.e. is the missing chunk of data too large to interpolate? - and thusly needs to be plugged with outside data.

2.1.4 Combining Map-Sets Together to Reduce Erroneous Terrain Data

Utilising OS as a base map resolution and combining with the LiDAR DTMs, then the DSM's realistic maps with improved accuracy of data points can be generated. When specific errors within the data-sets are found, procedures can be triggered to fall-back on procedural techniques, or gather input from a human.

The various maps of raw data and procedurally generated content can be visualised on top of one another to view the errors or differences between the maps. This is achievable with modern game engines; another reason to utilise computer game technologies.

The improvement combining of relevant geospatial data sets contributes to the knowledge of GIS data visualisation by the improvement of raw processed data sets. The contribution of algorithms specifically designed to combine and generate accurate geospatial data sets improve rendering of real-world virtual scenes.

2.2 Modern GISs

What is GIS? It is stated that GISs allow users to visualise a number of data sets relating to geographical locations[11]. GISs have many purposes for data; capturing, storage, manipulation, analysis, management, data-inference, and spatial depiction of geographical assets.

The Environment Systems Research Institute (ESRI) states that GISs' contain 3 main elements; Geoprocessing, Geovisualisation, and Geodatabase as depicted in Figure 4.

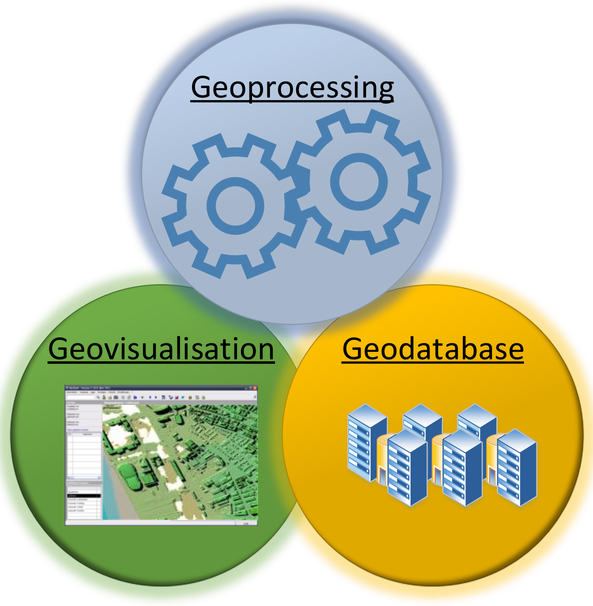


Figure 4 Components of a GIS according to ESRI.

Common datasets used within modern GISs are complex and varying, proving difficult for none domain specialists. Structured topological data, such as OS and LiDAR as well as descriptive geospatial data such as OSM can prove a familiar data set for none domain experts.

[11] http://www.esri.com/what-is-gis

Topology schemes must be stated to show the spatial relationships between assets within a scene. These topologies share many resemblances to network topologies, as they can be thought of as having similar structures, but represent different data. Both types of topologies still reference real-world locations. With multiple data sets being geotagged with real-world geolocations, the data-sets can be layered together. This is especially interesting for different visual textures such as aerial images, of varying types. A common list of layers can be; aerial satellite images, postal codes and boundaries, buildings and artefacts, user defined data, land use, or transportation. Within the OSM web portal, a user can view many types of maps, but the base images for a majority of the maps are grey-scale satellite images.

As stated, user defined data can be textures, layers, or visualisations depicting particular information such as; pollution, population, crime rates, noise levels, and many others. This is the main ability for which we believe modern GISs' need to improve upon and allow users to view scenes with customisable data views through encoding and binding data to spatial or visual parameters. Combining GIS data with modern CGT can be achieved using this.

GIS data is a complex confusing set of large volumes of data, varying by many types, and structures. The volume of data GIS process to create data for viewing is taxing on many computational resources. We wish to create our own internal processes to provide assistance and at the least improve data sets for visualisation. The processing of data, once an algorithm has been created, is not a significant problem due to the constant lowering costs of hardware, the problem is the creation of an algorithm which can depict and unravel those large volumes of data-sets with their particular specific relationships between each other. When working with real-world data, especially openly generated by untrained or trained users or volunteers, errors will inevitable be introduced to the data-sets. Procedures must be put in place to detect these errors. This in itself is a daunting task. How do we create an algorithm which can detect errors, when one does not know of the error we are looking for?

As stated by ERSI, compilation of data is an advanced and specialised technique. They issue the statement that geographic rules and commands are necessary to create correct and accurate data, but this is expensive. They state that this is why GIS data-sets are shared between companies and users. Although this is true, with that of TIGER giving their highway data-set of most of America to OSM. Currently, due to the expense of data capture, the reluctance to share has driven a wedge between advancements and collaborations. We wish to create algorithms to infer data from open sources to generate data which we are unable to obtain.

The obvious GISs which people think of first are those in the main stream news outlets; Google Maps, Bing Maps, and Apple Maps. This is partly due to their infrastructure, and their mobile applications being supported in many modern devices. These large corporations provide their assets, applications, and data to users through online web portals. Applications on smartphones and tablets can also be construed as online web portals, due to the data being pulled down from a server, just like a desktop web-browsing program. We will focus on the largest map provider, *Google Maps*[12], for our discussion of issues surrounding these types of GISs. *Google Maps* provides many different varieties of map data, in a large number of applications; *Google Maps*, *Google Earth*[13], and *Google Street-View*[14] etc.

2.2.1 Comparison of GIS Providers and the Data they provide

Before we discuss the differences between the different GIS providers and applications, we wish to state common functionalities with the GIS platforms, and what makes a GIS platform.

2.2.1.1 What is a GIS platform?

A GIS platform comprises of many separate systems interconnected to provide many services. Figure 5 is obtained from a release document from the ESRI website[15].

The platform shown in Figure 5 shows the combination of a desktop application and a lightweight web application. Modern hardware is providing advanced web support, and porting techniques used within desktop applications such as visualisations and 3D software interactions is capable, yet limited.

For our work, we will not utilise web technologies, but we acknowledge the potential need due to the wide scale usage of web technologies. DirectX or OpenGL within a desktop application, with direct access to the GPU, has the potential for further advanced visualisations.

Next, we present the high level functionalities and issues with many of the current GISs relating to this work and thesis.

[12] https://www.google.co.uk/maps
[13] https://www.google.co.uk/intl/en_uk/earth/
[14] http://www.google.com/maps/streetview/
[15] http://downloads.esri.com/support/documentation/ao_/698What_is_ArcGIS.pdf

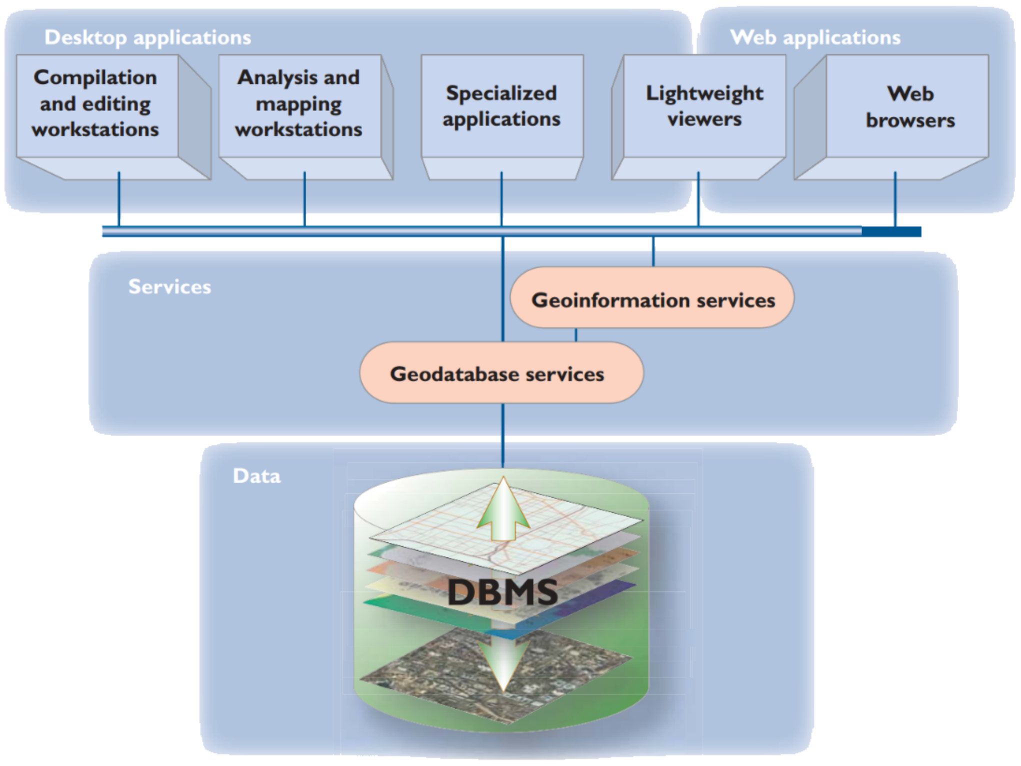


Figure 5 ESRI Comprehensive GIS Platform designed to facilitate geographic requirements. Image obtained from the document downloaded form 'http://downloads.esri.com/support/documentation/ao_/698What_is_ArcGIS.pdf'

2.2.1.2 Google Maps

Google maps is commonly known to the general public and provides access to automated navigations, and simple overlays of data, and can have custom elements overlaid if the license agreements allow them.

When a map is zoomed above a certain value, the user is allowed to rotate the angle of the camera (or viewing angle) in increments of 45°. The user can also arc/tilt the camera viewing angle downwards. This only allows a user to see an environment in a limited number of angles. Using a modern rendering engine, utilising multiple virtual cameras, a user can navigate an environment in any angle, arc, or perspective they wish. We wish to build upon this notion to allow a user to navigate an environment in whichever way they wish through the use of 3D camera systems commonly used within CGT. The integration and definition of 3D camera systems will be discussed later in this work.

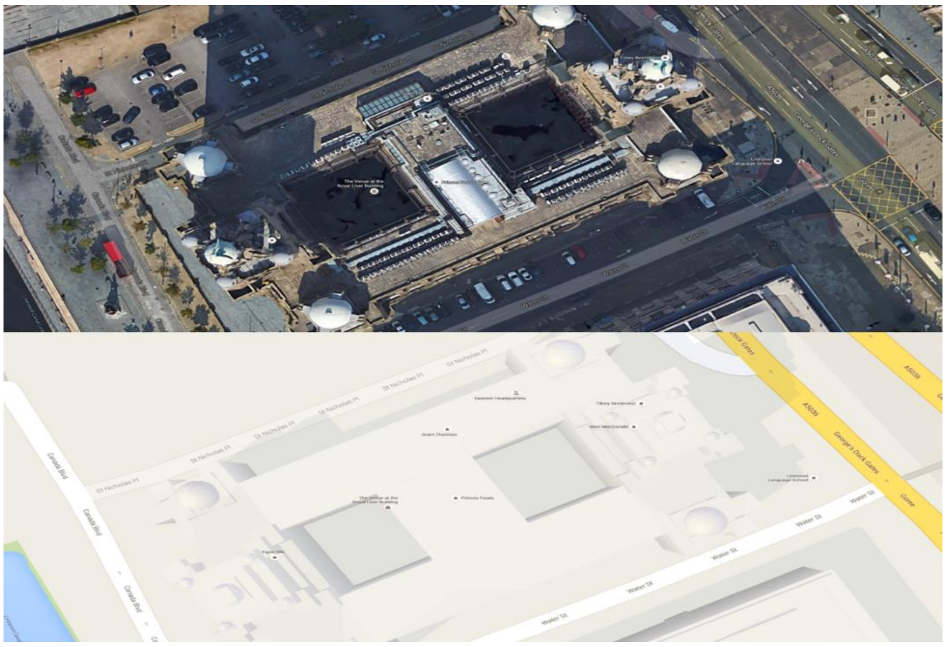


Figure 6 Google Earth and Satellite screen captures. Top has 3D building with images. Bottom is 3D primitive building with no textures. Image obtained from https://www.google.co.uk/maps/@53.405743,-2.9958568,19.25z

Figure 6, shows the un-textured 3D model of the Liverpool Liver Building (bottom), and the textured version (top). This is proof that a 3D approach to GIS is becoming more relevant and needed. The image also shows the overlay of highway data in the form of a grey strip and yellow strips to the right. They also have single node placeholders within the building, easier to depict in the bottom image of companies housed within the Liver Building. This data is also available within other GIS such as OSM. Although information is shown to the user within the web-browser, a user is unable to manipulate data in any form, apart from changing the visuals from satellite imagery to grey-scale form and camera angles.

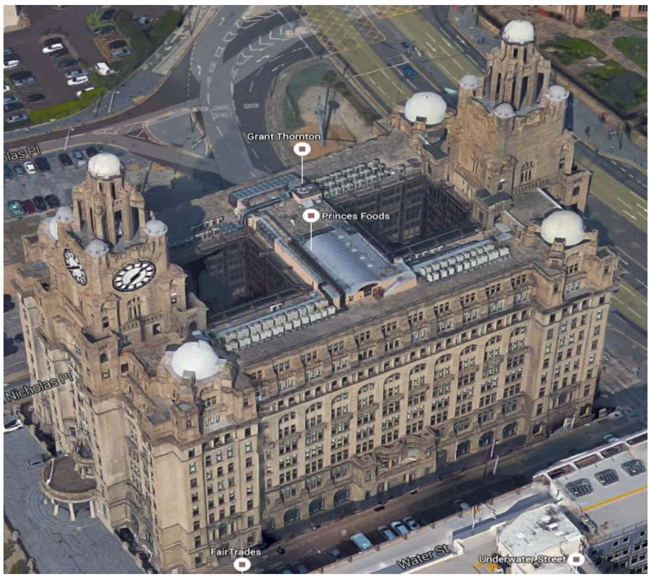


Figure 7 Google Maps 3D vision of Liverpool Liver Building. Image captured within running Google Earth.exe application.

Figure 7 , shows the 3D representation of the Liverpool Liver Building. This is an impressive representation of the building in question. As shown in Figure 8 and Figure 9, the procedure of generating this asset is error prone.

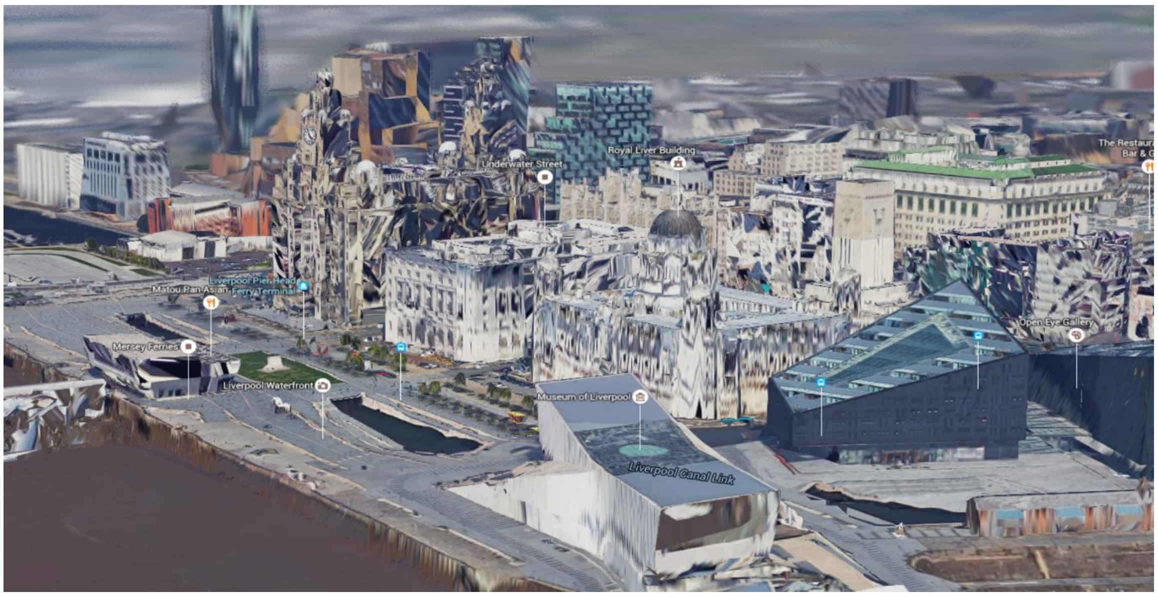


Figure 8 Google maps tilt of 3D city. Errors persist within image. Image captured within running Google Earth.exe application.

Within Figure 8, we can see the camera tilted to view a city landscape containing 3D buildings. After waiting a considerable amount of time, the errors within the image mapping persists within the scene. Users may not be able to recognise a scene. Having buildings pre generated and loaded from disk would remove the procedural nature of this view and texture splattering.

Another issue with the Google tilt view environment is the texturing of the 3D building assets. Due to the perspective and arc of the viewing angle, textures are used from their aerial 45° captures. These textures are interpolated across the building model mesh structures, creating a shearing effect which resembles an oil effect filter over the image. This removes realism from a scene; a realism which can be improved on with other games technology module pipelines; pipelines which utilise PCG. Procedurally generating building assets, and applying appropriate textures can alleviate this problem. We will need to use PCG techniques combined with additional data for the querying of specific building information. This is to determine the appropriate textures to apply to a PCG building with correct mapping schemes to reduce potential visual artefact errors, and to improve visual realism.

Google Maps utilise a technique of multiple Level-of-Detail (LoD) tile sets, which are loaded as needed within the multiple zoom levels. The tile sets are primitive, or low resolution images/data sets which are temporary assets which act as placeholders while the higher resolution assets are being loaded, or procedurally generated. The waiting time by the user is not large, and is becoming increasingly faster due to hardware advancements. Pre-processing GIS data can improve loading times by creating all levels of the LoD pre-runtime. This will increase visualisation times but may incur larger storage needs.

Map services provided by Microsoft, Google, Apple, and other private companies implement stringent legal restrictions with relation with their products and data. The data with the map providers can also be out of date, by up to 3 years. Figure 9 shows that the street view image was captured within July 2014. This is partly due to the expense of capturing aerial or satellite photographs, and also the fact that it also is a world mapping scheme and would need further funding to achieve up to date data and imagery. User generated content can alleviate this ‘out of date’ issue. For the reasons stated, we aim to use a dedicated platform for world mapping, with standard data generation procedures which allow user generated content to be added in real, or close to real-time. The need for a platform which allows for user generated data is the fact that the users of this framework we wish to research and develop, can input their own data constructs for visualisation. As well, data entered can work in tandem to improve visualisations for our work, but also for the platform itself. The obvious choice of platform is OpenStreetMap.Org. See the section on OSM later in this chapter.

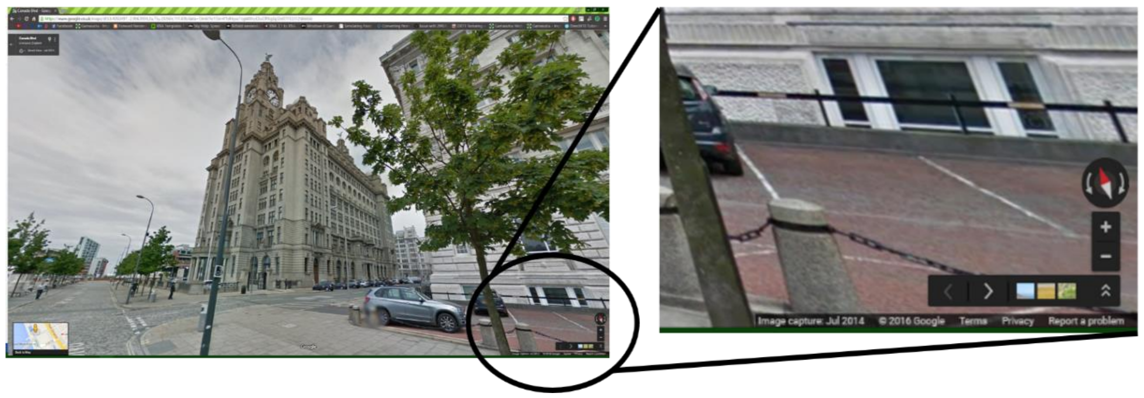


Figure 9 Google Map images capture date. Image captured from https://www.google.co.uk/maps/@53.4051016,-2.9964405,3a,90y,39.16h,89.41t/data=!3m6!1e1!3m4!1sQwIFrWOnhWhqmqqpqg6wlQ!2e0!7i13312!8i6656

Google Maps, and other map providers allow users to view multiple layers, in place of, or over the aerial images, or vector images. These layers are commonly; routing paths for vehicles, bicycles, and pedestrians, traffic data, terrain elevation data, place names, highway names, and others layers. These data layers are often unable to be combined within one visualisation, or rendering window. The use of advanced rendering techniques which are commonly utilised within modern rendering engines, will allow multiple layering of multiple maps, containing multiple rendering shader effect compositions. This will be discussed further in the implementation phase of this work.

Creating 3D GISs which improve on the 2D aspect need to be built from the ground up. Many navigation systems use 3D virtual scenes with procedurally generated simplified building models, and 2D projected alerts for speed cameras, or road side services. They also place these assets above simplified large polygonal terrains, or even flat planes. This is due to many reasons. The extremely large price for LiDAR data, which can be used as realistic terrain heights model meshes. As well as the lack of full coverage of LiDAR data at high resolutions for full coverage of the UK, let alone the world. The low resolution terrain meshes of 3D navigation systems can be made from OS data for the UK, which is provided freely by the UK government. OS data covers the complete UK. To create a higher resolution terrain mesh for use with an advanced rendering GIS will need complete terrain coverage of the area being visualised. LiDAR data is of high resolution, but incomplete for the UK, whereas OS is complete for the UK but of low resolution. LiDAR data being of higher resolution is a clear choice to use for a high resolution modern 3D GIS. The issue with it being incomplete for many areas is a problem which needs to be overcome. We cover further discussions and how to analyse and fix incomplete errors within LiDAR in a later section.

2.2.1.3 ArcGIS

ArcGIS is the most advanced visualisation and mapping project which relates closely to our work and research. The data and techniques which they utilise have been used by commercial projects and emergency services for the prolonged productivity benefits. This GIS provider provides data visualisations for many sectors; business, defence and intelligence, education, government, health and human services, mapping and charting, natural resource management, public interests and safety, and transportation visualisations. This is a large company with many years' worth of experience. Although they provide many simulations and visualisations of data overlays and layers, it is cumbersome to understand what is being shown. Partly this may be due to the lack of experience from ourselves, or it could be a design flaw by only allowing users to view data which they have processed already; data such as land value prices, and house prices which are advertised to the 'Business' sector to plan future expansions etc. It is this restriction of singular layered processed data which is simply added in a multi-layer visualisation which we wish to adopt and build upon. Providing a 3D view, instead of the glossy 2D overlays, will allow users to be comfortable viewing and understanding real-world scenes. ArcGIS can benefit majorly with the added open data provided by OSM. It is this open data which has extremely helpful, but over-looked data which can be useful for property developers. For example, if a business needed a new office and wanted views from its windows and be situated with large pedestrian traffic presence, utilising the current ArcGIS may not give enough information to provide the user with as much information they need. Using open data with positions of tree structures can add realism to a scene so the user knows that a tree will or will not cover the view from the building. Adding more data has multiple benefits; a user may not know they miss the data until it is presented to them, and they can always remove objects from a 3D scene if needed.

2.2.1.4 Additional GISs

The other GIS providers consist of; Microsoft Bing Maps, Apple Maps, and MapQuest. These providers are not as relevant as that of Google for its wide spread use, and its large cartographic data and images, nor are they as developed as ArcGIS in terms of data visualisation techniques.

2.3 Features of GISs

Within this section, we state the features of modern GISs researched from application web portals, research papers, and utilisation of the applications.

ArcGIS[16] website states the features of their application. The application utilises;

- Apps for Everyone – the application works on many devices; tablet, smartphone, desktop, laptop. We take not that an application/framework should work not only on multiple devices, but also, various ranges of hardware specifications.

[16] https://www.arcgis.com/features/features.html

- Ready-to-Use Maps – this represents that maps should be available for use or editing, or further development. We take from this that scalability is an essential feature for GISs.
- Visualisation – the visualisation of data, and big data is needed for a modern GIS. The utilisation of 3D and 2D rendering techniques, as well as functionality to apply multiple rendering techniques to single or multiple data sets for real-time rendering is needed.
- Analytics – the application states that analytics of data is utilised to make sense of the vast, varying data sets for the quantifying and impact of decisions made from the data. We take from this statement that some form of analytics is needed to improve accuracy of the original data sets, or infer information.
- Administration – ArcGIS states administration is a feature of their application. We state that administration is not needed for a modern GIS. If sensitive information is utilised, then administration is needed to restrict data access.
- Marketplace – ArcGIS application utilises a market place to share techniques, maps, and data sets. We take from this that scalability is needed, and plugin data is needed to be allowed within a framework. For example, components should be able to be placed within the codebase to extend the functionality of a GIS framework.
- ArcGIS Solutions – ArcGIS states a feature is solutions, i.e. premade maps with data associated with the map. We disagree with this implementation of a feature. This is a product and not a feature.
- Secure and Trusted – The utilisation of sensitive information needs security which is why administration is needed. We state that trusted data sets come in two forms; trusted provider, and complete and accurate data set. The latter is more important than that of the former.
- Tools for Developers – this feature references to allowing access to the API of the framework. We state that this feature is needed for open projects as our project is.

Other features from alternative data sets, GIS providers, and GIS applications are as follows;

- Multiple viewing angles, projections, and navigations of viewings. This can be implemented with the utilisation of CGT camera systems by allowing 3D scene navigation and the viewing of perspective and orthographic projections.
- Utilisation of satellite imagery, or OS imagery. We state this is a feature needed to combine previously used 2D maps, and reference the viewing inside a 3D scene. This allows transitioning to a new form of GIS framework.

- Highly detailed 3D model assets in the form of procedurally generated, and user input. Google Maps provide procedurally generated models, and also streamed model objects from web servers. They do not allow user generated content.
- Modify objects/maps in real time, running queries, selecting object, and modify the properties of said objects singular, or as a group.
- Animate objects to bring attention to objects. The animation of both spatial, and visual parameters should be a feature.
- ArcGIS, Google Maps, and other GIS application provide visualisations of data. The applications do not access the GPU of a desktop machine. This is a feature which CGT can provide and improve GIS visualisations.
- Viewing large scenes is needed as a feature. To complete this feature, LoD techniques are needed. This may be multiple model objects within a scene, or a data structure which can swop assets out in real-time to keep a steady framerate.
- Real-time framerates are needed to keep a user immersed.

In short, we conclude that the features needed for a GIS should include the following;

- 2D and 3D Camera system with multiple viewing projections, and controllable by user
- Using satellite imagery, and other 2D/3D content to bring realism to scenes
- LoD to improve large scene rendering.
- Real-time framerates
- Scalable development tools/API
- Allow importing of user generated content
- Realistic procedural assets
- Advanced rendering techniques applied to scene and individual objects
- Object selection and modification
- Geospatial data visualisation

2.4 Projects within the Domain of this Project

Similar projects to our work utilise real world GIS map data or a variant as a test beds for simulations. Within this section we will discuss these projects, and the similarities to our project. We analyse the projects on multiple levels; from their end visualisations, and the data they have used within the system. We also discuss which aspects we wish to include in our work, and which we intend to improve upon.

2.4.1 Autonomous Learning Agents for Decentralised Data and Information

Autonomous Learning Agents for Decentralised Data and Information (ALLADIN) [5]–[7] is a research project, in which the main investigation was into MAS and in particular: studies of data-fusion techniques, decision making, reinforcement learning, streaming data, resource allocation, distributed coordination, coalition formation, and decentralized control. The simulation is split into three domains; Situational Awareness, RoboCup Rescue, and Evacuation.

Artificial agents are out of the scope of our project, but we wish to improve on the 2D representation of the maps for which AI agents used in projects such as ALLADIN can be situated in.

2.4.2 SAVE

The SAVE (Sustainability Assessment Visualisation and Enhancement) project visualises the interactions between society, environment, and economics, as well as the future possible consequences of the current behaviours through mathematical modelling of materials used in construction [2] [1]. The simulation uses the Analytic Network Process (ANP) methodology [22], a network built of elements, each having individual attributes (numerical interaction values with the world), and creates a super matrix containing initial judgements of these attribute values, and then using a pair-wise comparison against a fundamental scale (basically a table of judgments of the priority of the elements) to create a comparison matrix which is used to create the eigenvectors of each element. The eigenvectors are combined with the super matrix to create an unweighted super matrix which is then calculated with a final pair-wise matrix representing the interactions of the clusters of elements and the eigenvectors to give the final weighted matrix. This becomes the measurable priorities of the sustainability factors corresponding to objects in a real world. This matrix can be applied to objects in a virtual world through shader programs to visualise colour changes applied to model meshes. This project is beneficial, and highlights the need to have functionality for users from many domains, setting properties within the system for the area they are interested in. What is meant by this; a user, who is concerned with the flammability of assets within a building, can set original scores to each model mesh which represent particular parts of the building. The framework then highlights these model meshes by utilising the techniques stated already, and creating an eigenvector representing a colour RGBA parameters, which is applied to the real-time rendering pipeline to manipulate, or modify the final visualisation of the rendered model mesh.

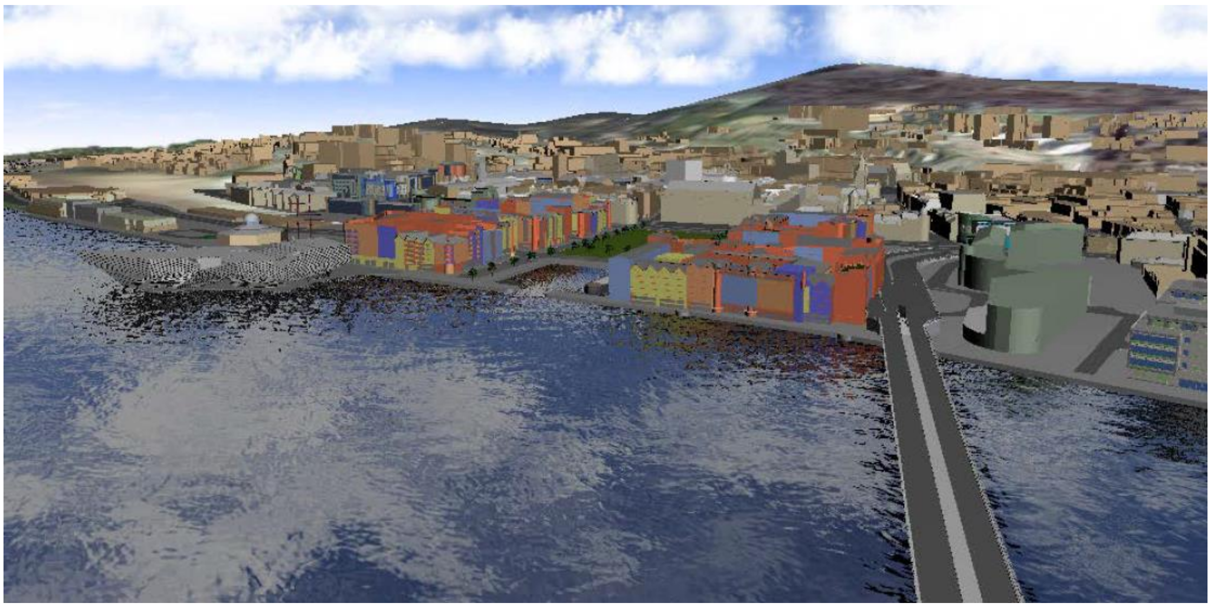


Figure 10 Image obtained from [2]. *Image shows 3D city with user generated colouring of buildings depicting sustainability options.*

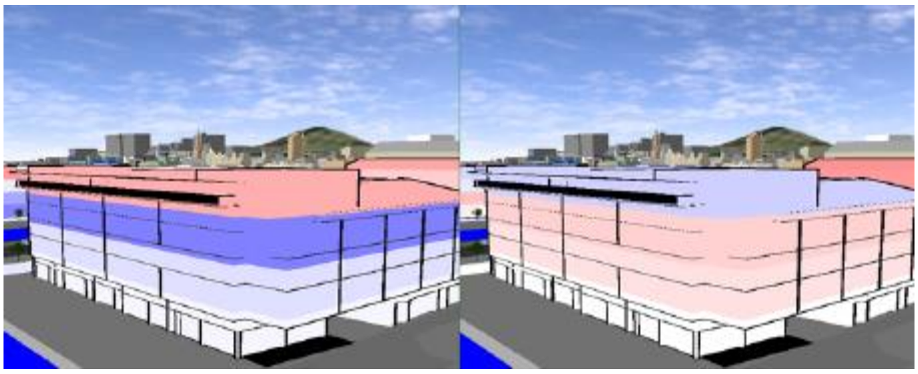


Figure 11 Image obtained from [2]. *Image shows multiple renders of a 3D building according to the sustainability of individual floors of a building. Each image shows varying matrix priority inputs.*

Figure 11 shows a building shaded using two different blending parameters. The left has a high environmental interest parameter, where the left has a low environmental interest parameter.

The project also introduces sustainability factors into other sectors in the form of individual projects:

- Sustainable River Pollution Management
- Sustainable Land Use
- Sustainable Regional Planning
- Sustainable Coastal Zone Management

We would like to implement similar techniques with our modern GIS visualisation system. Creating a matrix representation is an interesting, but advanced domain for the visualisation of user defined data. We suggest a technique, which is explained further in a later section, which allows a user to apply data straight onto objects within a scene, at runtime. Allowing users to visualise any data, on any part of a scene. We will state that this technique requires users to convert data values to a limited number of parameters which are obtainable through a CPU to GPU buffer object.

We contribute to the domain of GIS by applying the techniques of data encoding into algorithms, and data structures applied visualisation of object within large virtual scene selected by user generated queries.

2.4.3 Projects of Significant Scope

In the work by Fritsch and Kada [23], they state the parsing of the geo-data into a data format for use with a game engine was one of the largest obstacles they had. Their work concluded in visualisations of outdoor and indoor. Our work will not deal with indoor visualisations, only outdoor to begin with. Visualisation of individual floors of a building can be presented in a multitude of ways, taking into account the specific details of each floor.

In the work by Kada et.al [24] for the real-time visualisation of urban landscapes, they use open data for the generation and rendering of a 50km^2 area of Stuttgart containing 36,000 building models. Even though this work is over a decade old it is a high benchmark to achieve with modern hardware and software. They state that the simulation allows for interaction with the simulation, but the results state that the frame rates were between 2-11 FPS. We state that this low framerate is not acceptable for modern graphic capabilities and intend to reach at least 30 FPS with large scenes. They use a continuous LoD approach and implement an imposter technique. The imposter technique swaps out models for higher or lower polygon counts and sometimes billboards. The project allows fly-by and walk through animations, but lacks adaptation for a user. The user only sees what the creators of the simulation want them to see. We will improve upon this and allow manipulation of assets within the scene. Another issue with the work by Kada et.al is the lacking of additional information and visualisations. The project does not contain highway systems, or additional boundary data or amenity visualisations. Utilising the open data found within OSM, we wish also to include this extra data to provide a more in-depth representation of urban environments. We wish also to generate a procedural algorithm which can generate urban environments anywhere within the UK, to begin with, but adapt the framework so it can produce any urban environment within the world.

In the Master's thesis of Abhishek Sood from the faculty of San Diego State University, he uses the Microsoft XNA framework to generate the NEEVENGINE, an engine specifically designed for the generation of multiple types of serious games. Serious games, by definition, are games which are designed and generated to teach or train a specific subject. The objective of the work is to generate a flexible game engine for creating multiple types of serious games. The engine is generated with a modular architecture, which he states allows for fast prototyping. We will keep this in mind and consider a modular approach to our framework architecture.

A closely related Doctoral Thesis of Bo Mao, 'Visualisation and Generalisation of 3D City Models', has many similarities to our research, whereby the author explains the need for a modern GIS which integrates 3D objects for improved realism [11]. CityGML[17] is used to generate building model mesh data, rendered using X3D[18]. The work and research is built for mainstream web browsers. Due to the lack of infrastructure and rendering capabilities of modern web browsers, he employs a LoD technique which renders a scene in different scales; block, building, and building with façade to improve rendering speeds and scene realism. Shell representations of buildings are used which are generalisations of the higher polygonal building meshes. Textures of the façade are projected onto the building as separate polygonal quadrilaterals. These include windows, doors, and other facades. The textures are stored within compressed formats.

The compressed textures are used with UV coordinate mapping. This allows textures to be stored as texture atlases. A common technique with CGT but not of GIS or visualisation systems. Within this project we will not use texture atlasing but will consider the technique for future work.

The author utilises a LoD technique through their novel scenegraph structure, CityTree, which culls buildings which are not adjacent to the street for which the viewer is looking, while the user is in street view mode. This is a very interesting technique, but instances where taller buildings in the distance are culled prematurely; a situation which removes realism from a scene. Our framework will also cull objects in the distance, due to categorising assets to spatial 1kilometer squared tiles. This is a limitation in the work, and procedures need to be put in place to minimise the removal of assets in distances.

Within the published work of Bo Mao and Lars Harrie [25], they iterate the previous work of Bo Mao. They still conclude that a main problem with the use of 3D city models is the large volume of data. Within the paper they experiment with the process as whole, organising procedurally generated buildings models at a variety of levels-of-detail, in a scenegraph structure called 'BuildingTree' and then storing the individual assets within the NoSQL database for dynamic visualisation requests. For our work we will not utilise database storage for our prototypes but will store assets on hard disk as

[17] http://www.citygml.org/
[18] http://www.web3d.org/x3d/what-x3d

individual files for dynamic loading. The use of a database to store and retrieve the needed assets is a possibility for future work.

2.5 Commercial Influences and Game Simulators

Commercial products are created for entertainment and the production of money for the company. The popularity of simulator games has risen significantly in the past decade as stated by Techreviewer[19] and Gamasutra[20]. Popularity of commercial computer games, and serious games have also increased. We justify the discussion of such products here due to their relation to our project and the insights which we can obtain.

An ability which the commercial products utilise is the ability to manipulate the terrain. The interesting issue with the terrain is not the technique of manipulating the model mesh representing the area, but the ability to texture splat the terrain. Texture splatting is when multiple textures are applied to a model, normally terrain due to the high memory costs, and weights are used to state the blending or use between these textures. For example, stating terrain to be rendered with a grass texture with weight 0, and sand with weight 1, a user can change the look and feel of an environment by manipulating the value. More than 1 texture can be used to build a more realistic and varying terrain. If the slope value between two vertex points are within or out of bounds, then a rock texture can be used or a ground texture. This creates a procedurally textured terrain with cliff faces.

2.5.1 Cities Skylines

Cities Skylines, created by Colossal Order Ltd[21], is a commercial product which simulates the building and running of virtual cities. The complexity of this simulator is significant because they utilise a complex urban eco-system. Each decision the user makes has an impact on the whole simulation and city. The humans within the game are Artificial Agents (AAs) and each have their own needs, desires, and impacts on their local environments. Thousands of AAs can be running simultaneously and rendered at real-time rates. This shows that cities the size of real-world small cities can be simulated.

With each decision having an impact on the simulation, the user can view a scene in many different views; or data-overlays. Cities Skylines has 23 data-overlays: Electricity, Water, Garbage, Education, Citizen Happiness, Health, Levels, Wind, Traffic, Pollution, Noise pollution, Fire safety, Crime,

[19] http://www.techreviewer.co.uk/why-are-simulation-video-games-so-popular/

[20] http://www.gamasutra.com/view/news/188054/Whos_buying_all_these_niche_simulation_games_anyway_We_found_out.php

[21] http://www.colossalorder.fi/

Transportation, Population, Outside connections, Land value, Natural resources, Districts, Leisure, Heating, Road maintenance, and Snow.

Screenshots of these are found in Figure 12.

Viewing this type of data, but with real-world generated data is an aim of this project. We wish to make similar visualisations within our case study. The image on the top left and top right can be done in single pass rendering pipelines, whereas the bottom left utilises a multi-pass rendering pipeline. These techniques will have to be researched further to determine the justification of including such techniques within the PVS framework.

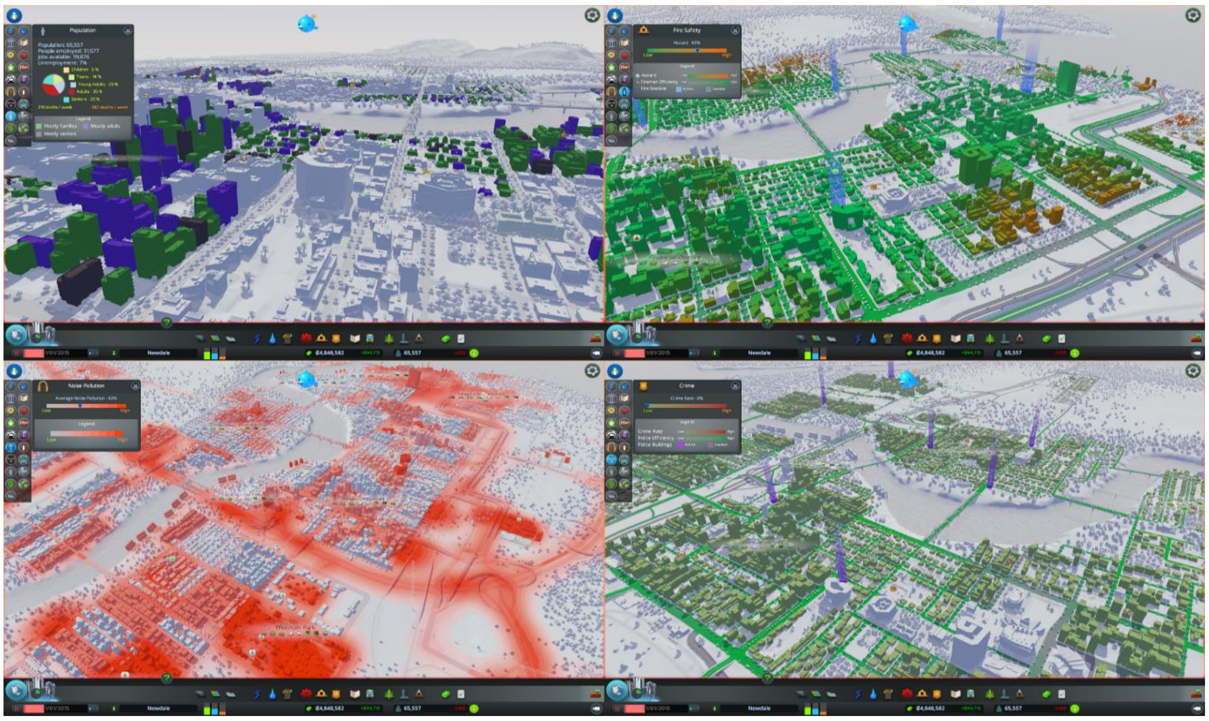


Figure 12 Cities Skylines[22] visualisation overlay screen shots. Top left is population. Top right is fire safety. Bottom left is noise pollution. Bottom right is crime.

2.5.2 Planet Coaster

Planet Coaster by Frontier[23], is a theme park simulator and rollercoaster building game. The reason we discuss this game which is currently in Alpha release is its use of Physically Based Rendering (PBR) techniques, and other advanced rendering techniques, for its in game assets. They model the assets separately and shade them according to their physical attributes such as plastic, metals, and concretes. They also utilise multiple lights within the scene and shadow rendering techniques. Combining the techniques to run in real-time is a difficult task but brings a high level of realism.

[22] http://www.citiesskylines.com/
[23] https://www.frontier.co.uk/home/

Utilising techniques similar within the PVS framework will bring improved realism, and advanced rendering capabilities to a framework of this type.

Figure 13 Promotional shots of Planet Coaster[24] by Frontier Games.

Another aspect which we wish to adapt within our framework, which is in Planet-Coaster, is the user navigation through mouse and keyboard. The user can select multiple cameras (First-Person, Orbital, and Free-view). These camera views seamlessly translate between the selected. This is an aspect of many simulations drawbacks, especially with real-world mission critical simulators. Seamless animations to different camera views, or to view a selected item which translates and orientates the camera to the coordinate's states adds additional information to the user's knowledge of a scene. If a camera was to simply snap to a selected item, then the user may become confused as to where exactly they are in relation to other artefacts within a scene; potentially leading to misjudgement and wrong decisions being made.

2.6 Urban planning and Maintenance

The creation of 3D visualisations of real-world locations has the capability of being used for urban planning. Research is underway for the generation of realistic urban environment visualisations, but much of it overlooks the potential for urban planning, the generation and visualisation, or depicting the interdependencies between infrastructure elements.

[24] https://www.frontier.co.uk/home/

By utilising real-world data to generate virtual cities, the properties of individual assets within the scene, which have been extracted or inferred from the raw data, can be used to visualise and categorise a scene for specific purposes and visualisations. Within OSM, a building can be tagged as fire-proof or not. Having a framework to allow a user to select and see all potential none fire proof buildings within a scene can provide great insight for emergency services as the fire-brigade. The fire-brigade and then position assets accordingly to prevent potential disasters.

Overlaying additional data-sets pertinent to the police force can achieve a greater understanding of large areas of urban environments. Combining maps with crime data, 3D heat maps can be created. *Cities Skylines* (a commercial simulation game) has this type of overlay visualisation within the application as shown in the bottom right in Figure 12.

The image shows mostly green which states that there is a low crime rating. The police stations are highlighted with vertical purple streaks. The highlighting of emergency building is available within the game. Utilising this technique with a modern GIS can achieve greater understanding of scenes through categorisation and visualisation of OSM objects.

2.7 Visualisation for Crisis Management

We have already stated the use of a modern GIS for highlighting of procedurally generated 3D objects created from OSM data to understand complex and dense scenes. This brings the discussion onto crisis management.

Visualisation of urban environments is crucial for disaster management. To represent real-world environments, realism is a key. User created building models as well as procedurally generated buildings are needed, as well as techniques to categorise data within scene using queries generated by the user.

By generating virtual scenes of real-world environments, disaster control and management can be undertaken. When disasters such as flooding, fire, or mass evacuation occur, they can be visualised and managed. A modern GIS using games technologies have the potential to be adapted for the visualisation and simulation of evacuation techniques by use of the terrain data we propose to use and the highway structures found within OSM.

2.8 Procedural Content Generation Definition

Procedural Content Generation (PCG) is the process of creating data using programmatic techniques. There are various versions of PCG; purely computer based, and taking user input to guide the process of procedural algorithms and generation techniques.

PCG is used heavily within the computer games industry, and is starting to be used within other domains of research and commercialism. The reason for this is the reduction of cost that computer processing brings to the domains and companies. For example, within the games industry, creating a single level of a game can take a team time to generate in a level-editor, to then have a paid artist to create the level itself. The speed at which the levels can be generated is directly correlated to the speed at which the artist can work. This is the same for a 3D modeller, and a 2D texture artist. Having programmatic procedures to replace the artists can greatly reduce costs, and time for creating assets and levels. The drawback of this technique is the knowledge, experience, skill, and time needed to create these algorithms which can produce what an artist can produce.

PCG has been used to create a large, varying palette of 2D and 3D assets, as well as complete virtualised worlds for use within modern computer games. The issue with the generated assets is the lack of realism. An artist can create an asset by sculpting, and modifying the object until a satisfactory result occurs. A programmatic technique needs to be iterated and tested vigorously before real-world deployment. Combining both artist (or user) and programmatic algorithmic techniques can increase production; the programmatic approach creates a base object which is generic, and then the user can either regenerate the assets but with various parameters modified to create a new base object, or modify the result by hand to create what is needed.

Creating a framework which can visualise real-world locations, generated from real-world map data needs procedural methods. The assets which are needed for real-world scenes are: vegetation, buildings, highways, terrain, waterways, human created assets (benches, post boxes, telephone boxes etc.).

The majority of assets can be procedurally generated, and vast amounts of open source software are available for the vegetation algorithms. Freely available software for procedurally generated buildings, highways, terrain, and the other mentioned assets are technically available (Unreal Engine 4) but do not take real-world data into consideration. We will need to create algorithms which can parse, and combine multiple real-world data-sets to achieve the project goals.

PCG is used already in many commercial projects such as Google Maps. Google uses procedural techniques to generate 3D building assets, and to procedurally cut texture images from their databases and wrap the textures onto the buildings. The reason the procedural techniques are used are for data-storage issues. It may be more beneficial to procedurally generated assets and data at runtime than to store said data or assets.

PCG techniques are utilised within computer game projects for the reduction of production costs. We iterate that these same issues are abundant within the GIS domain. CGT PCG techniques have overcome many issues with improved production times by offsetting artistic creation to algorithmic processes. CGT PCG does not utilise real-world geospatial data sets due to the size of the data sets requiring much resources which is not currently available within computer games due to other subsystems requiring processing (AI, quest systems, physics etc.). We contribute PCG techniques to offset processing of GIS data for improved production times, and generation of virtual real-world scenes for the domain of GIS. We also contribute to domain of CGT by the creation of PCG utilising real-world geospatial data sets.

2.9 Architectural Models for High level rendering

This section introduces game engines, and their usage within a project as we propose. The engines in question will be a small selection of all game engines available. Each will need to be carefully chosen for their many different functionalities, attributes, and productivity requirements. The learning curve into deciding which engine to utilise must also be taken into account for fast prototyping and final artefact development.

Functionality	Unity3D	UnrealEngine4	XNA	MonoGame	Cocos2D	CryEngine 3
3D Support	1	1	1	1	0	1
DirectX 9	1	1	1	1	0	1
DirectX 10	1	1	0	1	0	1
DirectX 11	1	1	0	1	0	1
DirectX 12	0	0	0	1	0	1
Input / Output	1	1	1	1	1	1
Keyboard/Mouse	1	1	1	1	1	1
2D Support	1	1	1	1	1	1
Forward Rendering	1	0	1	1	1	1
Deferred Rendering	1	1	0	0	0	1
Middleware Support	1	1	1	1	1	1
Screen System	1	1	0	0	0	1
Audio	1	1	1	1	1	1
Windows OS	1	1	1	1	1	1
Mobile Support	1	1	1	1	1	1
Open	1	1	1	1	1	1
Free	0.5	0.5	1	1	1	1
Networking	1	1	1	0	1	1
Touch Screen	1	1	1	1	1	1
OpenGL	1	1	0	1	0	1
Mac OS	1	1	0	1	1	1
HTML5	1	1	0	1	1	1

Table 2 Game Engine Functionality Comparisons

Within Table 2; blue represents that the engine has the capability of said function with a value of 1, red represents that it does not have the functionality with a value of 0, yellow represents that it has the ability but under certain circumstances with a value of 0.5. Some blue represents pseudo ability. The pseudo ability in this case is the 2D support. The engines do not have support for 2D, but create the illusion of 2D by generating 3D assets but only projecting them within a 2D world; i.e. the Z plane (depth) will always be set to 0.

Table 2 is also organised in order of value for which the functionality is needed, from highest to lowest need. 3D support is having the highest need, while HTML5 support has the least functionality need. The use of 3D rendering is needed for 3D virtual scenes. Frameworks which showcase geospatial data are commonly shown within 2D projects; ALLADIN, Google Maps, Satellite Navigation Systems etc. 3D project will improve realism and engage users further.

The project revolves around the rendering of real-world scenes. The utilisation of the DirectX rendering APIs allow for complex and realistic rendering calculations. DirectX 9 has a higher ranking than that of DirectX 10, 11, and 12. This is due to technologies which utilise DirectX 9 rendering pipeline have already been implement, and thusly can be utilised within our proof of concept work.

Forward rendering is ranked higher than deferred rendering due to its functionality and ease of creation. Deferred rendering does allow multiple lights within a scene, but also decouples scene lights from objects.

The latter functionalities; screen system, audio, mobile support provide future support for future research avenues. Network functionalities provide additional interaction with the proposed prototype framework by allowing multiple users to utilise a single simulation instance.

HTML5 support is a functionality which would greatly benefit this work by providing visualisations of real-world environments through web browser support, but the current implementation of HTML5 will not support this projects, thusly a desktop application is preferred. HTML5 is included as a future needed functionality.

From Table 2 we can see that the majority of commonly used engines all have use of common functionalities and thusly the use of any of these game engines can be a good choice for creating modern games and modern GIS.

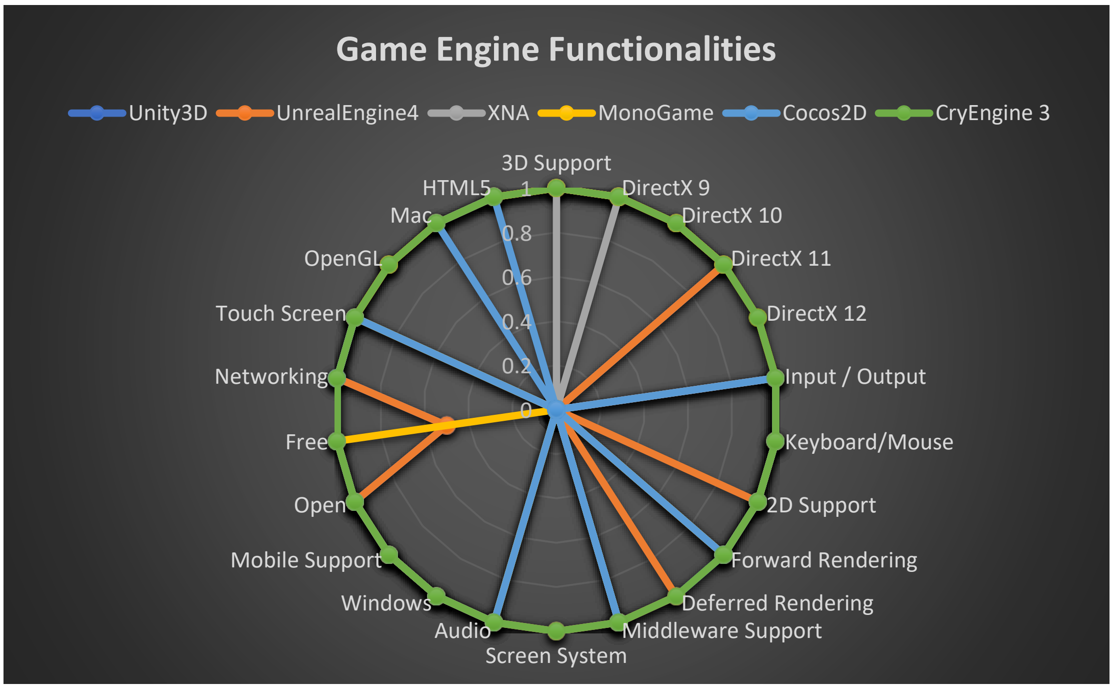


Figure 14 Spider Diagram of Game Engine Functionalities

The graph in Figure 14 shows that the CryEngine3 contains all functionalities needed for our project, and for potential future work. Notice that the proceeding MonoGame engine of XNA has additional functionalities which may benefit a framework of this type.

Unity3D is a common engine, but restricts use to a C# language. For a real-time rendering system C++ would be a better choice due to its low level memory management which if implemented correctly will prove beneficial to rendering times and techniques. For this reason, UE4 is the better choice than Unity3D. UE4 utilised Blueprint technology and additional functionalities which prove beneficial for rapid prototyping. The issue between these are the licensing issues and potential money pit for the future publication of our framework.

Microsoft has ended support for XNA, and although it is still useful as a rendering engine capable of large scene visualisations, the limited support is an issue. MonoGame is the open-source successor of the XNA framework. It allows porting from XNA to MonoGame framework with minimal of code changes, but sadly, as of writing this work, MonoGame is currently not fully implemented to a standard which XNA is. MonoGame does allow DirectX12 rendering techniques, whereas XNA is restricted to DirectX9 and simulated CPU generated DirectX10 techniques. As a proof of concept project, the use of XNA is still a valid choice. The benefits it provides by allowing MS-Build capabilities is functionality which other engines do not provide. The MS-Build allows precompiled of assets used at runtime simulations and rendering.

We contribute CGT, specifically that of computer game engine, to GIS for improved and advanced rendering of real-world virtual scenes.

2.10 Rendering Techniques

When discussing rendering techniques, we must introduce and compare rendering techniques commonly used within CGT and GIS. Next we discuss individual rendering techniques commonly used within visualisation systems and CGT. We introduce possible benefits and drawbacks of the techniques. They are not ordered in any specific order due to the complexity of implementations of the techniques.

2.10.1 Forward Rendering

Forward Rendering [26], [27] is the technique of rendering objects in one at a time.

Creating the illusion of many lights can be achieved by using static light maps, or baking lighting calculations within a scene. This is a technique done by applying additional textures to a scene which has a predefined light image on it. Think of a car in the dark with its lights on. Having a single car with 2 dynamic lights as headlights can be done easily, many car objects with multiple lights would tax resources and processing power. Early artist got around this issue by creating 2D light ray texture and combining the texture with a transparency blend function within the shading technique, the texture gave the illusion that lights are on within a scene. This would greatly reduce lighting calculations to a single draw call and a blend function.

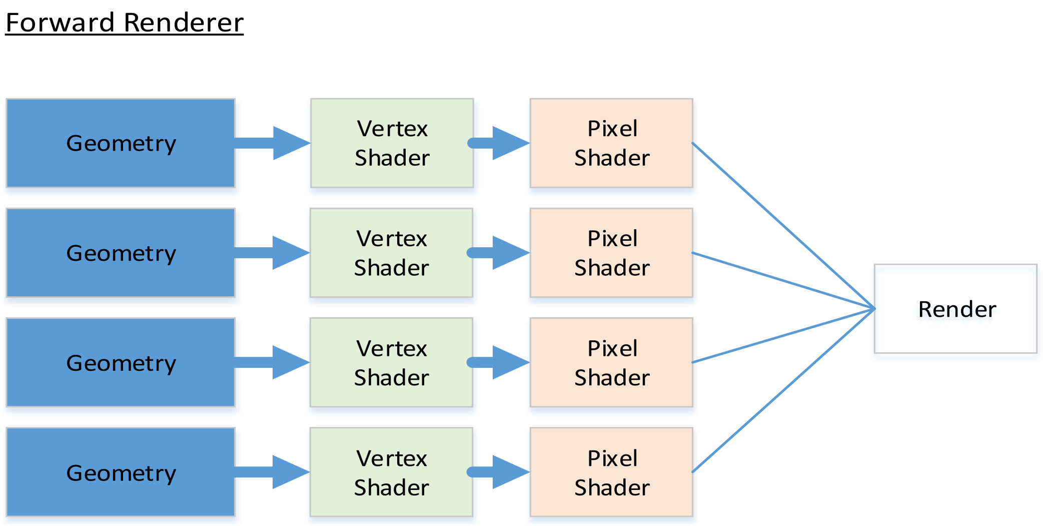


Figure 15 Forward Renderer Process

2.10.2 Deferred Rendering

Deferred rendering [28], [29], as the name applies, leaves rendering to a later stage. The process splits rendering into two passes; a geometry pass, and a lighting pass. The geometry pass renders the geometry, particularly the individual properties of geometry into render targets; position, diffuse colour, normal etc. A render target is a buffer which stores data for later use. Normally a texture buffer is used in this case, and the data is encoded within the RGBA pixel of the texture. The decoupling of the geometry rendering from the lighting calculations gives the ability for a large number of lights to be applied to a scene.

The algorithm in simple terms creates a bounding box or bounding circle around point lights, and it is these bounds which are checked against the stored pixel buffers and used to light the area of radiance from the light. If using spot lights, then cone or pyramid bounds are used.

Deferred rendering can be used within city models by creating many lights within a scene. These lights are instantly thought to be used with street lights. They can also be used for any point or spot lights within a scene such as emergency vehicular lights. Deferred rendering may not be the best choice due to its issues and drawbacks of flexibility.

Issues with deferred rendering consist of; needing a GPU which can generate multiple render targets, and contain high bandwidth parameters due to the passing of the large buffers of data. Transparent objects cannot be used unless specialised algorithms have been implemented. This is normally platform and application specific due to the effects it has on the rest of the rendering pipeline. Using forward rendering for the transparent objects only can alleviate this problem.

A major problem is the use of multiple materials within a scene. Normally a single material is used. Multiple materials can be used but this requires a large amount of video card memory to store the large number of render targets. Another issue with deferred rendering is the inability to use anti-aliasing techniques which are commonly automatically applied within the rendering pipeline. When using deferred rendering, the results of the anti-aliasing produces incorrect results. This can be overcome by using edge detection techniques and blurring the results. Modern techniques utilise post-processing techniques such as Morphological Anti-Aliasing (MLAA), Fast Approximate Anti-Aliasing (FXAA), Subpixel Reconstruction Anti-Aliasing (SRAA), and Multisampling Anti-Aliasing (MAA) [30]. We will utilise techniques which are built into the game engine which we decide to use.

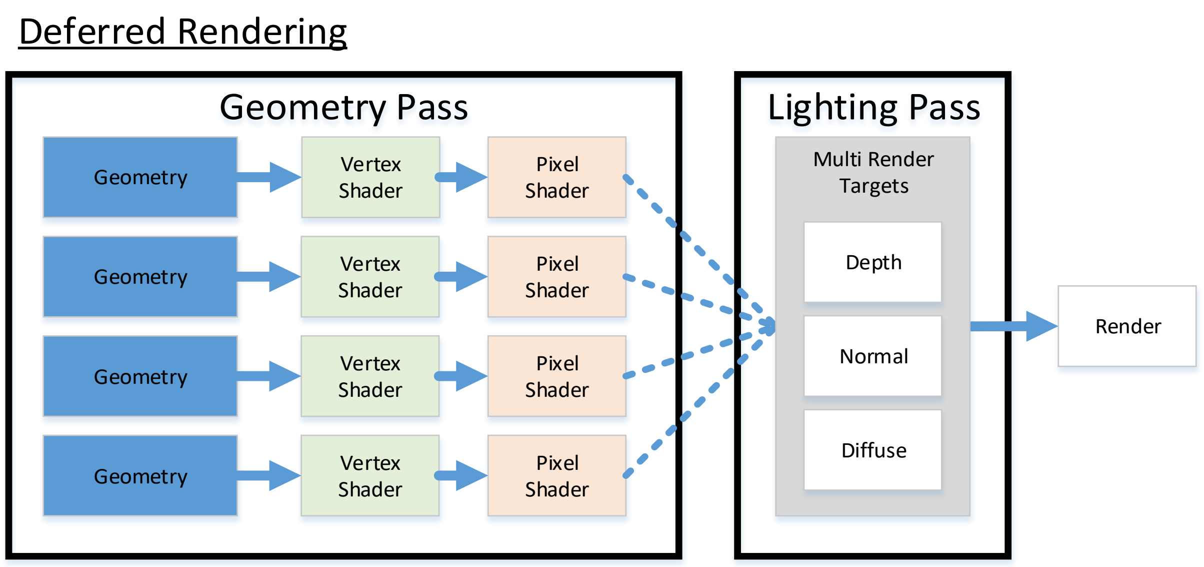


Figure 16 Deferred Rendering Process

2.10.3 Deferred Lighting / Light Pre-Pass

Deferred lighting or Light Pre-Pass is a modification form of deferred rendering. It adds an additional pass than deferred rendering, going from 2 passes to 3 passes. The first renders the geometry to a buffer containing the Z depth and normal's. This data is used for the lighting equations. The second renders the attributes of diffuse, specular, and material properties. The third pass re-renders the geometry and reads in the material properties combining with the lighting results, outputting the final per-pixel shading values. This technique greatly reduces the memory footprint by the reduction of buffers, but at the code of rendering each model twice. Within certain situations this technique is faster than deferred rendering, but in other deferred rendering is faster.

Deferred lighting can be used with MSAA on DirectX9 hardware, something which cannot be done by deferred rendering. This is appropriate due to the relevance that DirectX9 still has within rendering technology.

2.10.4 Conclusion of which Rendering Pipeline to use

The right renderer depends on the project itself, and the results for which is needed. For complete realism then ray tracing would be the preferred method but current technology restricts the use of ray tracing within real-time parameters. If using many dynamic lights within a scene, then deferred rendering or light pre-pass may be a better choice than forward rendering.

To summarise, the use of appropriate rendering techniques applied to GIS will contribute to overcoming the issues with current GIS renderings and visualisations. CGT implements many options of rendering techniques. We will contribute these options to GIS. The use of advanced rendering option of IASF contributes to GIS to allow multiple rendering options applied to single assets, or large scenes.

2.11 Spatial Data Organisation Techniques – Scenegraph

The need to organise large data sets into smaller groups for ease of processing to increase speed of algorithm completion as well as implementation of novel search techniques is paramount to a project working with data sets on a global basis.

A tree contains nodes, and connections between those nodes. The nodes can contain explicit data such as values, or from a high-level contain references to objects. The connections between the nodes are commonly called the edge. Using the edges multiple types of trees can be generated; a linear, acyclic, cyclic, directed, and undirected.

We will discuss the uses of acyclic directed trees with a parent/child relationship. Utilising these types of trees, recursive search techniques can be utilised.

When separating spatial data with tree structures, they are commonly called scenegraphs. A scenegraph is employed in many GISs and computer games for the organisation of spatial assets within the virtual worlds. A scenegraph lends itself to a framework we propose because of the benefits and ease of advanced search techniques, and algorithmic exploitation which can be applied to a whole tree, branches of a tree, single nodes within the tree, or leaf nodes. Each node structure can contain a reference to its parent, and have reference to its children.

With spatial trees such as a scene graph, each node is positioned relative to its parent; where ever the parent is translated, rotated, or scaled, the child will use this data and update itself. This is not true for all types of scene graphs but is true for the ones which are not are custom implementations. Using the translation, rotation, and scale a World matrix is generated which is used by the graphics pipeline to render the model mesh data normally contained within the spatial node to screen. We will not go into detail about World matrix generation here, but state that the ordering of the translation, rotation, and scale multiplication with each other has varying results. Multiplying the translation, rotation, and scale will result differently is multiplying scale, rotation, and translation. The latter process is called SRT order and is used within many computer games and visualisations. This order lends itself for creating visualisations which can draw a user's eye to a branch of the tree. Manipulating the scale parameter of a parent, all children will be scaled as well, highlighting objects within a scene making them larger.

There are many scene graph libraries other than *OpenSceneGraph* such as *FreeSG*[25] and other codebases from popular code sharing web domains as *GitHub*, *BitBucket*, *and NuGet* etc. A short evaluation of open scene graph implementations can be found in [31].

The scene graph libraries presented are a good starting point to understand the abilities of a scenegraph but for the framework proposed, and the organisation of large amounts of data, both for processing and visualisation, a custom scene graph structure will be needed. Experimentation into finding the right balance between ease of use, ease of implementation, search capabilities, and benefits a custom structure will bring is needed. The scene graph will need to be custom in layout, splitting large scenes up into manageable smaller chunks for processing. For example, the root of a scenegraph can be the *World* or environment being depicted, the next node a country, the next child node can contain all buildings within that country. See Figure 55 for a visual representation.

The structure of a tree used within the proposed framework must allow modification, and allow future search algorithms to be used with the tree structure. The tree structure in question must be open enough to act as multiple types of tree structures; directional, bidirectional, acyclic, cyclic. This will ensure that future usage of the framework and adaptively will be justified.

To use a scenegraph for a modern GIS framework or game engine, it must be mutable, and provide the user to initialise multiple search techniques; Depth-First, Breadth-First, Iterative etc.

To summarise our contribution, we wish to apply a scenegraph technology, which is commonly used within CGT, to spatially organise, and categorise geospatial data. The use of scenegraph technologies overcome issues of GIS data organisation, and scene traversal techniques for object querying and data dispersal. The contribution allows scalable virtual scenes, and a base platform for further algorithms to be implemented from.

2.12 Conclusion

Within the Background discussion we have introduced the domains needed for the generation of a modern GIS. We have presented the map data available and initial discussions on how the data will be used within the framework and research. We have also discussed areas of interest within the domains and already implemented commercial and scientific projects: SAVE, ALLADIN, Cities Skylines, and Planet Coaster. We have discussed different popular GISs which resemble our research and framework proposal very closely; Google, ArcGIS. We then introduced the standards of game engines currently available. The section on game engines highlighted that there is not a game engine which is deterministically better suited to be used for generating a modern GIS. Some provide

[25] http://freesg.org/

various implementations of rendering systems, each suited for varying types of visualisations as discussed. Forward rendering is suitable for rendering artefacts which include transparency, deferred rendering is best suited when more than an estimated 3 lights are needed within a scene. Ray tracing and Ray casting rendering systems are stated to be too advanced and unrealistic to work in a real-time parameters needed for a modern GIS. To organise large scene and to visualise them within this framework, we discuss the use of spatial and data organisation techniques, specifically the need to pre-process large maps into manageable areas. A novel spatial-graph structure used to organise the data is needed. The scenegraph will need to include multiple segments to organise the data not only spatially but also data dependently.

For our project we will utilise forward rendering and restrict the number of lights used within a scene.

The next section presents the literature review of our work, critiquing others research which is similar to our own and within the same domains as our research.

3 Literature Review

This section covers the related work of other scholars and researches. Due to the large domain coverage of our work, this is the reasoning to split the segments into separate chapters of *Background* and *Literature Review*. These chapters will reference each other and be partially mixed.

Figure 17 shows our research and experience with the current commercial and open GISs. From the specifications of functionalities, we state is needed, we notice that none of the GISs allow direct access to the GPU, nor do they allow custom GPU programmes to be applied to the data within the GISs. Providing access to the GPU will allow improvements to visualisations by tapping into the hardware acceleration of modern GPUs.

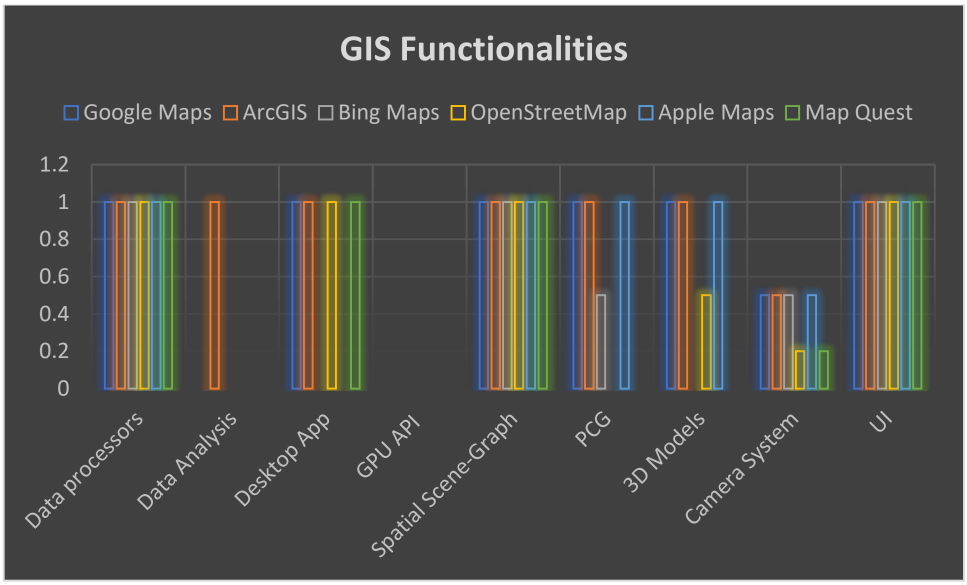


Figure 17 GIS Functionalities implemented from the specification we state is needed for a modern advanced GIS

3.1 Why use a Game Engine?

Within the work of [32], Baare and Grigg compare the productivity between game engines. We take from this work the speed at which the productivity took place from the experiments they undertook. They stated that the game engine of Unity3D was 1.9 times faster than that of using UnrealEngine4s' Blueprint mechanisms for development purposes. The Blueprints mechanism we speak of can be thought of as a 'drag and drop' component architecture used for rapid prototyping of commonly used assets/components/games types. Although the work was comparing the two game engines against each other, it emphasised the abilities of modern games engines which contain many user interfaces to commands, and the additional functionalities the game engines provide the user. They issue the statement that C++ knowledge should be known. This is not a knowledge trait expressively needed within modern computer game development; especially with the use of the drag and drop components of Unreal Engine 4 Blueprints and the use of other scripting languages such as Boo or JavaScript in Unity 3D.

For our project, knowledge of programming is needed to develop novel data structures, algorithms, and mechanisms.

Herwig and Paar [33] state the suitability of using a games based API for terrain and landscape visualisation. They discuss what problems game engines can overcome in relation to the visualisation of landscape virtual scenes. They state that 3D game engines require fundamental knowledge and expertise in 3D designs and level construction. They state working with geo-terrain data is commonly a trial-and-error approach for visualisation when loading into a game engine. We agree with this in regards to the large amounts of geo data available in many different formats that a considerable large amount of development is needed, but a trial-and-error approach is a derogative approach. We do however agree with their statement that game engines are a low-cost alternative than GIS and CAD software. For this reason, we will utilise game engine API which is either free to use, or has an open source licence.

Within many commercial game engines, scenegraph structures are implemented but restrict the data attributes attach to each node. Moa [11] creates a scenegraph to organise the buildings within his virtual city. He uses an open source scene graph implementation; *OpenSceneGraph*[26]. Combined with X3D[27] he generates and organises large amounts of data to be viewed within real-time standards. This proves that custom scenegraph data structures, specifically designed and built for virtual urban environment organisation and visualisation are viable. Although scenegraph structures have shown to be a viable solution to organising a city's worth of data, a novel data structure is needed to apply attributes from the more complex OSM data.

An example of a rendering API which can be utilised for academic research as well as game engine development is the Microsoft XNA API.

The XNA API has been used in many other projects for visualisation, training and many other domains. Ouch and Rouse use the API to generate a mobile driving training game to teach uses safer driving skills [34]. Denninger and Schimmel [35] utilise XNA for the teaching of complicated paradigms. They state the reasons to use XNA is for language simplicity being C# and not C++ which is difficult for beginners to understand. XNA is also a wrapper for the underlying DirectX making it a managed DirectX API. XNA has also been used on a similar project to this in [36]. Herrlich et.al. use

[26] http://www.openscenegraph.org/
[27] http://www.web3d.org/x3d/what-x3d

CityGLM data to produce virtual scenes. Assets are loaded in via COLLALDA, a mesh representation. We are unsure as to why they utilise COLLALDA as XNA has built in model processing capabilities.

Alternatives to the XNA API is JMonkey. At the time XNA was maintained by Microsoft, it was the more powerful of the two APIs according to the results of [37]. From the results, JMonkey has faster frame rates with low numbers of objects within a scene. Increasing the number of objects in a scene, XNA becomes the faster of the two. At 50 objects, JMonkey ran at 598.41 FPS, where XNA ran at 235.29 FPS. With 4000 objects, JMonkey ran at 4.95 FPS, whereas XNA ran at 29.13 FPS. XNA can still handle 16,000 objects at 4.69 FPS where JMonkey cannot. For this reason, XNA is still the better choice due to the expectation of large scene visualisation.

To summarise the contributions of our intended work, the utilisation of game technologies can fill the gap of knowledge and current issues within the domain of GIS, issues such as procedural content generation, and visualisation. These contributions can enhance virtual scene visualisations of current GIS standards.

3.2 Visualisations

In this section we introduce the current research within the domain of visualisation and concurrent domains which visualisation techniques and technologies have been applied to.

Robinson and Shapcott [38] attempt to improve data-mining understanding from non-domain experts of analysts by introducing visualisation techniques. They state to improve data-mining understanding that visualisation needs to;

1. Support the whole of the data mining process.
2. Are usable by domain resident data miners who are not expert analysts.
3. Support data miners who are from a diverse range of backgrounds.
4. Allow interaction with the data mining tool.

Remove the domain of data mining from the list stated and the statements are current for the domains of this project.

They state that visualisation needs not only bar charts or graphs but visualise relatable objects, and be directly integrated into an interface for the user to interact with as well as manipulate the scene or data. We agree with this. Having relatable objects, and manipulating them in many manners within a simplified user interface is needed for a modern GIS. The relatable objects need to be objects which many individuals encounter on a daily basis; buildings, highways, paths, amenities, transport routes and many other interconnections within modern cities.

Visualisation techniques can be used for example, to portray potential failures within critical infrastructures. Wilde and Warren [39] discuss how visualisation techniques can be applied to critical infrastructure protection and to visualise potential un-foreseen domino failures. They specifically focus on the interdependencies of internal infrastructures and of interconnected infrastructures. The paper is a research description of a wider research project but states the use of visualisation techniques to explain the characteristics and complexity involved in critical infrastructure protection and interdependency connectivity. This reinforces the need for a modern GIS which can visualise multiple data-sets within a real-world visualisation simulation. It also reinforces that visualisation techniques can be applied to portray complex interconnections and data to users. We will create a framework which can visualise 3D Big-Data city models, and allow the interaction from users to manipulate individual objects within a virtual scene spatially and visually.

Generating 3D urban environments lends itself for project managers and urban planners, both private and governmental, to advance the understanding of familiar and unfamiliar territories. The work by Falconer et.al. [1] does exactly this. They produce a virtual environment of a city within the UK and allow users to view the scene by procedurally generated visualisations with each component of the visualisation corresponding to a specific criterion the user is most interested in. The project is discusses in section 2.4.2. The components of the visualisations are parameters which the user can set values for e.g. a user can state the importance of price for components of building should be rendered from white to red, lowest value to highest value respectively. We will utilise this technique of applying attributes to rendered objects within a scene which are set by a user. The user can assign their own schema to the visualisations; creating the same effect to the visualisations as the SAVE project. Visualisations do not and should not only be colour based, and animations both spatially and colour based can provide an additional layer of information. For example, increasing the scaling of a building dependent to depict the value of a building, bringing the attention of a high valued building to the user.

Urban planning is a complex activity with many unforeseen issues. Modern hardware and software can alleviate the traditional methods of urban planning and be used for the visualisation of potential developments within urban areas. These visualisations can be used to encourage residents within an area to be on-board with the idea to change. This notion is discussed in [33], [40]. Our project can be utilised for micro and macro urban visualisations. Details of areas will need to be encoded as attributes or from lookup tables for each of the assets within the visualisations we propose; buildings, highways, amenities, etc. for this we will create data structures which will extract attribute data and create 3D model mesh objects which can contain this data. These attributes can then be utilised by a user to query the objects within a scene. The generation of a queryable algorithm for use with various types of objects is needed. We plan on using the .Net Language-Integrated Query (LINQ)[28] API for this task.

In the work of [41], the visualisation of cascading failures within critical cyber infrastructures is discussed. The framework provides the understanding of possible cascading failures within the networks, combined with the organisation of large amounts of GIS data. Combined to generate further understanding of the topological considerations of real-world network structures to provide awareness with past and future decisions.

An early prototype application built from our work has been utilised for visualisation of real-world urban environments for use by tinnitus sufferers, projecting procedurally generated 3D primitive model mesh objects created by data-driven techniques with real-world noise levels, section 1.4.1, item 4. The use of the prototype has shown the scalability and adaptiveness a game-engine can bring to visualisations.

Visualisations of urban environments, and the accompanying techniques used can be found in [42]–[44].

Visualisation techniques can be implemented with game engines for visualisation of scientific data as stated in [23]. Visualisations of other data types can be found in [38], [41], [44]. The wide domain of where visualisation techniques have been employed to portray complex and confusing datasets is encouraging.

The framework proposed will need the ability to adopt all the visualisation topics of the stated research. This is justifiable due to the similarities between the domains. With small changes and adaptations of the framework, plus additional data, the needed visualisations can be achieved, if not improved on by utilising the rendering techniques we propose.

Visualisation techniques for disaster management are also a domain which can benefit from a modern advanced GIS. Within the work of Kwan and Lee [45] they examine the potential of using real-time 3D GIS for GIS-based intelligent emergency response systems. Their framework allows the potential for the reduction of delays within multi-level structures. The framework also supports pathfinding algorithms between emergency service buildings such as fire-stations, and the disaster sight, taking into account traffic speeds, blocks, and stops. The framework is a fine example of what can be done with real-world data for 3D visualisation of cities for data overlay and data

[28] https://msdn.microsoft.com/en-us/library/bb308959.aspx

management. It proves that the framework which we have proposed is achievable in relation to visualisation technology for displaying data to improve the understanding of real world data maps.

To summarise, the issues of rendering and visualisations within the domain of GIS's can be alleviated by employing the algorithms and techniques currently used within CGT. We contribute our work by the implementation of advanced rendering techniques applied within the domain of GIS.

3.3 Spatial Analysis with Visualisations and 3D Models

Lv et.al. [46] present a big city visual analysis platform based on Web Virtual Reality Geographical Information System (WEBVRGIS). Within the platform there are extensive analysis functions which a user can select; spatial, terrain, sunlight, traffic, population, and community. The purpose of WEBVRGIS is to create a dynamic visualisation of geographic information, transforming GIS data into a more intuitive, and user realistic 3D experience. The analysis functions are intended for use by external domain experts. We wish to include spatial analysis techniques into the framework but for use within a data-inference algorithm which will calculate specific attributes about assets within a scene; e.g. calculating a buildings height from OSM if only the number of floors are present within the attribute description. Using a pre-defined value for the average height of a floor, the height can be estimated, improving data visualisations.

Utilising spatial analysis techniques can be combined with 3D generated cities for the creation of interesting and novel pathfinding techniques used by a multitude of vehicular structures and sizes. For instance, pathfinding for a person on foot will be different for a car, which will be different for an oversized vehicle which will need roads larger than a given width.

Pathfinding techniques commonly used by AI systems within computer games are spatial analysis functions. Many pathfinding techniques are available as stated by [47], [48]. We wish to highlight path finding techniques used for finding paths for multi-sized agents (vehicles, none player characters, water flow, etc.) through 3D virtual scenes. Harabor [49] present clearance-based pathfinding, a technique which generates paths for multi-sized vehicles within real-time strategy game Command and Conquer. The algorithm takes into account the destructible buildings, which when changed will update the 2D grid map representation used to generate the clearance-based A* path. This technique can be used with virtual scenes generated from real-world data to generate input maps for the clearance-based pathfinding technique. Using LiDAR and spatial analysis functions, a similar result could be generated for real-world users and real-world vehicles of varying size. A form of this is presented in [50] by using explicit corridors to generate obstacle avoidance paths. We are currently not utilising the framework for this purpose. A framework which can use geospatial data for the input for AI algorithms has not been implemented within the domain of GIS.

Hagelback and Johansson [51], [52] use potential-fields for creating input maps for pathfinding algorithms. We highlight this work because of its similarities with using textured LiDAR maps; converting height points to pixel colour values of the 2D texture can give similar results to that of potential fields. Utilising 'depth rendering', a technique to render objects according to the distance they are from the 3D camera, potential fields, or height maps can be created. We plan on utilising this shading algorithm within the framework. This allows for further visualisation combinations and potential avenues for future work and research.

Utilising texture formats, we discuss in section 6.8 a technique to store all height data within the UK within a novel storage technique within a formatted texture file by employing parsing procedures within the algorithm.

Within the body of knowledge, the use of spatial analysis techniques applied to real-world data sets to infer information. The contribution of procedural techniques to analyse and query the data for the generation of additional data for the improvement of visual accuracy of real-world virtual scenes.

The next section discusses the literature and techniques of Procedural Content Generation (PCG).

3.4 Procedural Content Generation

To introduce PCG we refer to the work of Khaled, Nelson, and Barr [53] who present metaphors commonly used with PCG. PCG has input into many domains and thusly various names are used for the same techniques.

Many of the projects, which create 3D city representations use a form of PCG for the creation of data or visual assets [36], [54]–[58]; buildings, highways, terrain, etc. Muller et.al. [55] present their procedural modelling techniques used to generate realistic, theme inspired, buildings. We state theme inspired because their technique can be tuned to create buildings which represent various characteristics of well-known and distinct cities, new and old. Figure 18 shows a procedural representation of the ancient city of Pompeii. We show this image from the paper [55] because we wish to highlight the realism it portrays. The human eye is adapted to spotting patterns within scenes, especially scenes which represent real-world instances. Errors, or repeating patterns are easily spotted. Within [55] each building is different in some way, be it textures, rooftop styles, shape of the building, and the height of each building. This is achieved through the use of rule-based comprising of 190 manually written shape rules. What is great about this work is the fact that each buildings blueprint is accurate from archeologically documents. Using shape rules to create, and augment a building is an interesting technique to generate realistic, but procedurally devised buildings.

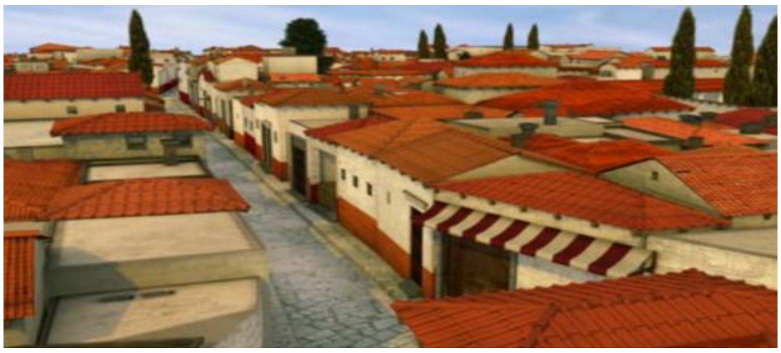


Figure 18 Screen grab of a procedural representation of Pompeii from the work of Muller et.al [55].

Games set within real-world parameters: Friberger et.al [18][19] use open GIS data to create interesting variations of well-known games; Monopoly with real-world place names, Top Trumps with real-world countries with accurate states, and using real-world maps to create 2D tile maps for a Civilisation style game. The interest with this work is not the end-results but the intuitive nature for which the games suggest; taking real-world map data and integrating that map data within popular implemented commercial games.

PG of buildings for city models are discussed within the SIGGRAPH course proposal [59]. Within the work of Parish and Muller [55], [60], they discuss their techniques for generating 3D buildings by extruding primitive polygons by multiple heights, as well as attaching primitive model meshes (cubes, rectangles, and most commonly tubes) to create complex and realistic buildings. They also discuss the techniques for attached facades to the buildings. Within the research, the facades have effect on the building structure and vice-versa. Their work also includes generation of highway structures commonly found within real-world instances, but are procedurally generated not taking into account of real-world data. The generation, like the building generators utilise rules for particular instances of error/issue to resolve constraints. The highway generation utilises a self-sensitive L-System to great success.

Our work will not need the use of L-Systems for the highway generation due to highway data already described in the OSM data. Our work proposes to implement procedural techniques to generate highway structures, parsing from OSM data to 3D model mesh data.

A survey paper discussing the procedural techniques for city generation is found in [61]. They present the work of [62] who use primitive polygons for the combination and generation of floorplans which are then extruded to create buildings. Some polygons are extruded at different levels. Using a similar technique combined with OSM building boundary data, we can generate extruded buildings. Reducing the scale or the boundary, we can create roof tops for the buildings, just as [62] have.

Their work takes into account the population density, creating highly dense building structures and road systems to where the population is higher. This creates high-rise buildings and a limited number of large road ways between the populations which act as separate cities. This technique works to great success and produces city landscape similar to real-world cities. We are not interested in procedurally generating large cities, but procedural techniques are needed to generate objects and assets within a scene. Procedural estimations are also needed to calculate estimate parameter values for said objects; the number of floors within a building. Knowing the number of floors within a building can improve visualisations by texture mapping accordingly.

Evaluating the realism of this work with the results of the proposed framework can take place. With a procedurally generated city made of real-world data and comparing the results of a procedurally generated city which is generated by pre-defined terrain layout and rules is a starting comparison point for our work.

Martinovic and Gool [63] utilise Bayesian Model Merging but within a 2D textural domain to resample and generate stochastic façade textures to replicate buildings with the same style. Resampling images of real-world facades and locations is an interesting approach to create none-repeating city structures. As stated the human eye is very adapt to spotting repeating instances. An issue arises within this instance of increased memory footprint, or if the data is procedurally generated when needed then extra processing is needed. For our work, this would be a future parameter of the work needed. Currently we will focus on the use of advanced IASF and the user ability to integrate additional data within the visualisations.

A survey paper discussing search-based PCG is found in [64]. Togelius et.al. discuss many features of PCG and we wish to highlight some of them. They discuss the need for either pre-processed or runtime content generation; what needs to be done before the application is used by a user or can be done in the background as the user interacts with the system. They state it is done pre runtime, then manual editing of the final artefact can take place which cannot be done at runtime generation. We agree with this and stipulate that PCG of assets such as buildings/highway/terrains should be done pre-runtime. Other content which can be done procedurally at runtime may be debugging features such as creating the bounding box mesh objects to view the spatial scenegraph bounds we described earlier.

The use of seed values for PCG is one of the benefits of creating large assets but only storing a single seed value. Togelius et. al. discuss the differences random seeds versus parameter vectors [64]. An issue arises with the generation of this seed value. Creating complex 3D structures which are unique in nature and still classifiable to be 'correct' is difficult. Trial and error techniques are normally employed with these algorithms. The preferred method for commercial companies is to combine procedural techniques with manual editing done by artists. The artist will generate multiple versions of the asset, and pick the one they think is best and work manual from thereon. The alternative to this is to create a parameter based algorithm which takes into account the parameters listings. With procedural city generation techniques, this could be to limit the number of buildings, or limit the height of the build or the textures used. Although the writers of the survey paper split sections of 'Random seeds versus parameter vectors' and 'stochastic versus deterministic generation', we state that although they are vastly different, they also overlap vastly. Utilising a seed value to create assets is a form of deterministic PCG, and is a form of data compression. Stochastic techniques can be thought of as the process to create the assets in the first place. Within a city generation framework, the deterministic approach will be the best approach, combined with parameter based inputs for the PCG algorithm. This will prove efficient in terms of development as well as flexibility.

Combining this with the addition of manual editing is also needed, but not for prototypes of the framework but for the final product.

The framework of Galin et.al [65] creates procedural highway structures such as roads, tunnels, and bridges by analysing terrain data and descriptions of their geometric characteristics. Their research is still in the early stages in this field. The cost function to determine the trajectory of a path is calculated taking into account the slope of the landscape. Their results are positive and realistic. Combining this with real-world data, future planning of highway structures can be calculated accurately.

3.5 Photogrammetry

Photogrammetry techniques can be utilised for the generation of many GIS data-sources. In the work of [66], Enem et.al utilise photogrammetry to determine the height of buildings from a multitude of angled aerial images; top-down, 2.5D, and 3D perspective images. They state that in spite of the horizontal and vertical accuracies of photogrammetric maps, it is not possible to use them directly in 3D city modelling. Current techniques introduce errors which have to be edited manually by a data analyst specialist, or a 3D modelling artist. Thusly photogrammetric map images need to be processed further to create accurate 3D building blue-prints. Their system still uses manual editing of errors, but is one of the most advanced 3D modelling GIS we have seen during this research. Although the data produced, and in which produces procedural buildings, the realism is lost due to the non-textured and simplified extruded building structures. Another weakness within this work is the lack of additional assets within a 3D city modelling GIS. They only utilise 2 tree models, and 1 street lamp. We wish to improve on this by not only adding additional model assets, which is not a novel technique, but utilising other mapping data sources such as OSM to gather further information into the style of city being generated and the placement of additional assets; post-boxes, vegetation, bins etc. This extra information can be utilised by not only the visual inspection and realism of a scene, but also used by council and private officials. Post office workers can analyse the routes taken to gather assets, and see cause and effect of altering postal routes.

According to Dhonju [67], building outlines can be determined more accurately than that of LiDAR and aerial laser scanning. This is due to two issues. The angle that the laser is shot from the aircraft added with the angles and orientations of a building’s roof or sides. This issue can add noise to the data. And low resolution laser techniques in relation to the distance between lasers, not capturing close enough to a rooftops edge. We agree with these issues, and with LiDAR data which we have analysed, this is a large problem with LiDAR, especially LiDAR data with is captured at low resolutions. We present initial techniques later to reduce this error.

The use of photogrammetric methods is determined to be out of the scope of this research due to the benefits in which modern LiDAR data provides.

3.6 LiDAR

To counter the claims of Dhonju [67] about photogrammetry, within the paper of [68], claims that LiDAR is rapidly replacing the photogrammetric approach. We add to this claim with the research of [69] Montoya et.al. whom use LiDAR equipment attached to drones. This removes the huge cost of capturing aerial imagery needed by photogrammetry. Due to this low cost, data capturing can be completed on a smaller budget, and can capture high resolution contours of buildings and other real-world artefacts; something photogrammetry techniques currently cannot do.

LiDAR can be accurate to a few centimetres when captured from 800 meters above ground. Using building locations from OSM, we can extract the height of said building from the LiDAR data-set, doing away with complex photogrammetric procedures. Combining the two data-sets can bode well for each; the LiDAR is used to create height values for the OSM building, while the OSM boundaries of buildings can be used to check the accuracy of the LiDAR data points.

LiDAR has been used within many analysis, and visualisation projects such as the project stated in [70]. Within this paper the authors use Mannings' coefficient of ground texture and roughness to determine water flow for flood control. Using LiDAR data removes the need to visit sites in person, proving the relevance that LiDAR can bring to crisis management, visualisation, and scene analysis, and giving an insight into real-world environments.

Errors within LiDAR capturing is presented in the survey paper [71]. The multiple authors present the errors within different domains which contain the use of LiDAR. The errors stated within our background research of the use of LiDAR, and within the later sections discussing the issues of big geospatial data. The errors within the LiDAR data is an issue for our work, and this is why we introduce procedural algorithms to remove the errors.

3.7 Conclusion

Within this section we have discussed the relevant projects and research of others which our work can either improve, or we can use as a comparison for evaluation, or as a starting point for our own research. We have covered the use of game engines and why to use them for this project, and the research and uses of visualisation, specifically the frameworks and projects which are using visualisation techniques to view real-world big data-sets. We have discussed the standards of PCG in relation of GIS and visualisations of real-world locations. We have also presented issues surrounding photogrammetry and the techniques which have been used for. The successor, and popular alternative to photogrammetry is LiDAR which is cheaper to produce and accurate to a few centimetres when captured from 800meters above ground, and a few millimetres when using handheld equipment capturing technology. This proves well to create realistic virtual scenes for a modern GIS.

To summarise the issues stated within the literature review, photogrammetry has issues which can be removed by the use of LiDAR. LiDAR has issues and errors which can be filled by the use of OS data. OS data has issues due to low resolution and thusly the need for interpolation techniques and algorithms are needed to increment the low resolution to a higher resolution to combine with LiDAR to create a complete terrain data set. This is a form of PCG. Other forms of PCG have been utilised for building generation of Pompeii, but this work has issues. The issues can be overcome by the utilisation of real-world modern data describing properties of the assets being procedurally generated, i.e. buildings and highways. Visualising these assets within large scenes, scenes as large as a modern city has issues with rendering within real-time constraints. Techniques and algorithms are needed to overcome these issues. Techniques such as, LoD, advanced rendering categorisations, and scene organisations. The SAVE project utilises Eigen vectors to encode user specified interests. This is encoded into effect parameters, but are done offline. We iterate the need to allow this functionality at runtime.

The literature reviewed in this section have all been single instances of algorithms, or techniques. None tackle the issues within a single framework, processing the datasets to produce accurate real-world model sets for visualisation and interactions.

The next section covers the big-geospatial challenges. The chapter delves deeper into the data parameters and issues with each of the geospatial data-sets we have decided to utilise.

We wish to include spatial analysis techniques into the framework but for use within a data-inference algorithm which will calculate specific attributes about assets within a scene; e.g. calculating a buildings height from OSM if only the number of floors are present within the attribute description. Using a pre-defined value for the average height of a floor, the height can be estimated, improving data visualisations.

4 Big Geospatial Data Challenges

This section will cover all the difficulties and issues with data we propose to use. Multiple types of data available from multiple providers; government and private sector will be discussed. At the end of the chapter we discuss our final decision to what data we will be utilising and how it will be used in conjunction with the spatial data structures and algorithms we propose.

4.1 Ordnance Survey

There are multiple data sets relating to OS which vary slightly provided by the UK Government, and associated third parties. These variances are problematic when combining these data with other data sets such as LiDAR e.g. the sea level point for which all other points are referenced from are different for each data-set. The base sea level point is called the *datum*.

The multiple maps we have are; Digimap DTM, Profile DTM, Panorama DTM, OS Panorama DTM, OS.

Digimap DTM has issues being 4 to 6 centre meters' higher that the LiDAR data available. Profile DTM has an additional point of data for each column and row, resulting in duplicated data, but may improve model mesh generation techniques through filling the space between maps. Panorama DTM and OS Panorama DTM also have this issue. The metadata for each of the map types are different to the LiDAR data available, and thusly, additional processing is needed for which ever map type chosen.

With the OS map type, additional data checking shows that the height points are within 1cm of error with the LiDAR files we wish to use. The fact that the data is freely available is another benefit.

The properties of the map types are shown in Table 3.

Map Type	Data points in map (column and rows)	Cell Size	Area Coverage
Digimap DTM	200^2	$50m^2$	$10km^2$
Profile DTM	501^2	$10m^2$	$5km^2$
Panorama DTM	401^2	$5m^2$	$2km^2$
OS Panorama DTM	401^2	$50m^2$	$20km^2$
OS	200^2	$50m^2$	$10km^2$

Table 3 Ordnance Survey map variation comparisons.

The OS mapping scheme needs to be discussed further. This is for the issue that if all of the OS mapping scheme is to be mapped, as a consequence, the number of maps, and data storage, would be very large. If mapping for the complete UK at a resolution the same as the highest resolution of the LiDAR maps we have (2000^2 data points at 25cm cell size), a large data storage problem occurs. See section 4.3 for further details.

The mapping of OS can be thought of as layers. The lowest resolution of OS names each tile of area coverage with a single prefix letter; layer 1. Each tile represents $500km^2$ area coverage. Figure 19 shows the coverage.

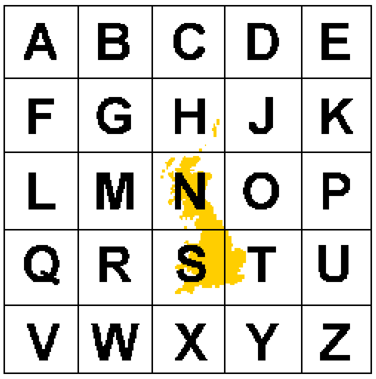


Figure 19 OS lowest resolution single prefix mapping scheme. Each tile represents a square 500k area coverage.

Layer 2 tiles represents 100kilometers squared area coverage and splits the layer 1 tiles into another 5*5 grid and they are named by inheriting the name of the tile which they are in, and utilising the same prefix as before, e.g. tile S in N would have the name NS.

Layer 3 tiles represent 10kilometer square area coverage and splits the layer 2 tiles into a 10*10 grid. Each of these tiles is named from the bottom left to the top right tile with respect to the easting and northing (or the X and Y coordinate).

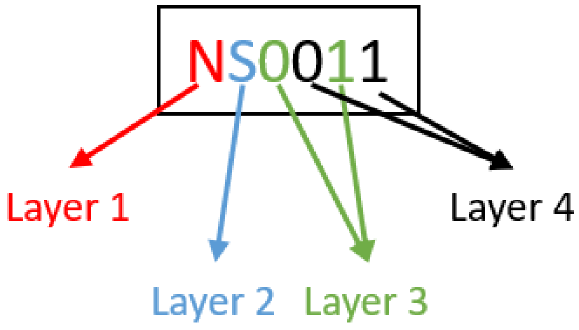


Figure 20 OS Reference scheme layer naming explanation for a 1kilometre tile location.

Layer 4 tiles represent 1kilometer squared area coverage and splits the layer 3 tile into 10*10 grid and adds an additional easting and northing naming parameters.

This is the end of the official OS reference scheme, but we can further this by increasing the easting and northing values. Extending the layers will incur projections in base 10 format. See Figure 21 for a visual explanation.

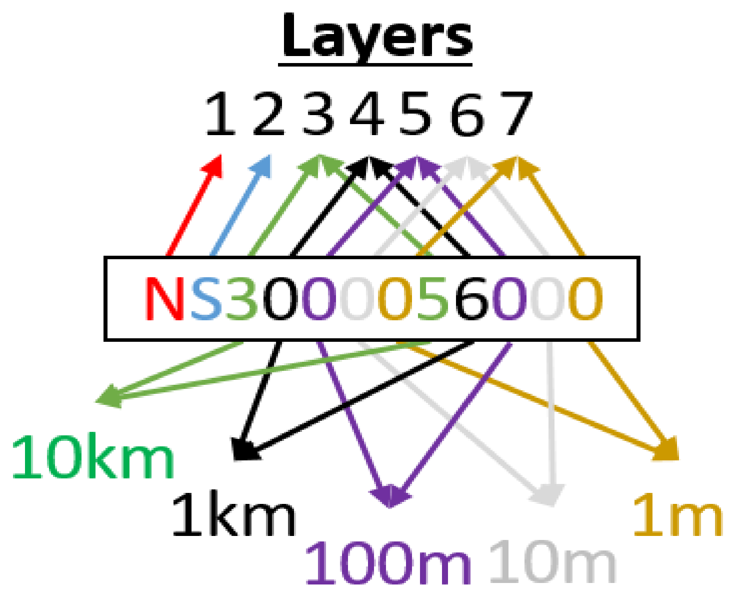


Figure 21 OS Layers and corresponding Distances.

An issue is the fact that the OS map files are only released at 10km^2 area coverage. Additional processing will be needed to extract and then interpolate the maps to the same resolution as the LiDAR maps.

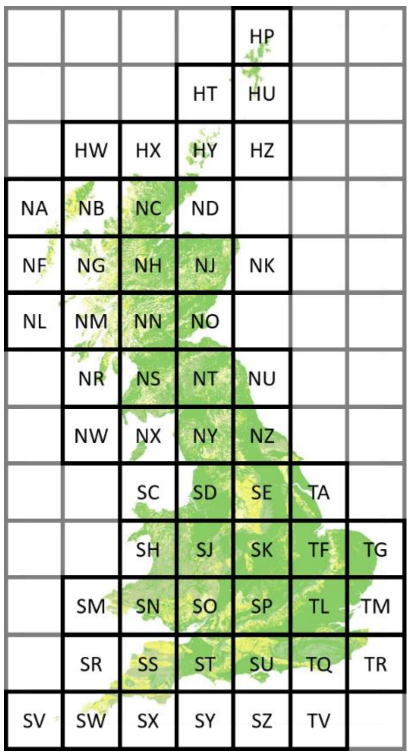


Figure 22 OS double préfix resolution. Each tile has a double prefix and represents 100km²

The total number of tiles in the multiple resolutions is stated next;

- Layer 1 500km^2 = 25 tiles
- Layer 2 100km^2 = 25 * 25 tiles = 625
- Layer 3 10km^2 = 100 * 25 * 25 tiles = 62,500
- Layer 4 1km^2 = 100 * 100 * 25 * 25 tiles = 6,250,000

The total number of titles within the mapping scheme is equal to 25 + 625 + 62,500 + 6,250,000 = 6,313,150.

Many of the tiles from each layer cover only sea. From Figure 22, only the shown 100km^2 maps can be made or processed. This will have improved processing time. The maps shown in Figure 22 equate to 55, giving 550,000 tiles at 1km^2, a reduction of ~=5.7 million tiles. We state 550,000 tiles represent land coverage, but as Figure 22 shows, not all areas of the tiles are mapped, only the yellow areas shown are actually mapping with OS data, thus further reductions can be achieved.

4.1.1 Ordnance Survey Conclusion

Given this information of the multiple data files available, the OS data provided by the UK Government is the preferred data format to use. This is due to a multiple of reasons; freely available, minimal metadata differences, row and column numbers are devisable by 10, as with the LiDAR data-sets. This proves well for extracting subsets of the map which can be interpolated to the higher resolutions of the LiDAR maps. The maps will have to be pre-processed to either introduce additional metadata separately in an individual tool, or introduce the metadata during the processing and combining toolset.

4.2 LiDAR

Regarding LiDAR data, there are many issues which may affect their accuracy.

LiDAR data is currently incomplete and inaccurate for the UK. It is unlikely it will ever be complete for the UK; high cost of capture, ever changing urban environments.

The LiDAR terrain errors are classified as; missing data, false data capture, and errors produced in initial processing from the LiDAR providers. Missing data is when data cannot or has not been captured. This can be caused by vegetation or water for which the laser used does not reflect of the object correctly. False data can be captured from dynamic objects; vehicles, flocks of birds, swaying trees etc. Any errors produced from the provider cannot be verified due to none-disclosure agreements.

LiDAR data can be provided by a number of companies and within a number of data formats at multiple resolutions. The data which we have obtained has been issued from the UK Government.

We can confirm from contact with the providers of the LiDAR dataset available, that the process contains a combination of: Optech LMS Manager[29], Bentley Microstation[30], Terrasolid[31] / Terrascan[32], QT Modeler[33], ArcView 3.3[34], ArcMap 10.1[35], and many bespoke tools for each of the pieces of software stated to create the DTM. This shows that processing LiDAR is a complex and processing intense procedure. The many steps may incur errors, which are processed through the pipeline. Additional checks should be made; algorithmically, or by human interaction.

Stated in the book [72], the authors states that LiDAR produces improved results of data capture in daylight hours. The issue with this is if capturing in daytime hours, multiple moving artefacts are more frequently found within the data-source.

The data parameters for the LiDAR data-sets provided by the UK Government consist of;

- Coverage: 98,255km^2 (65%) England and Wales (81% of all urban areas)

[29] http://www.teledyneoptech.com/index.php/product/optech-lms/
[30] https://www.bentley.com/en/products/product-line/modeling-and-visualization-software/microstation
[31] https://www.terrasolid.com/home.php
[32] https://www.terrasolid.com/products/terrascan_versionhistory.php
[33] http://appliedimagery.com/
[34] http://www.esri.com/software/arcgis/arcview
[35] http://www.esri.com/software/arcgis/arcgis-for-desktop

- Resolution: 2m, 1m, 50cm, 25cm
- Vertical accuracy: 5cm – 15cm
- Type: DTM and DSM
- Version of data-set: 1998 to present
- Formats: ASCII and Geo-referenced JPG
- 50cm resolution data is typically flown at 850M altitude above ground
- 1M resolution data is typically flown at 1000m altitude above ground

Another issue with the LiDAR data is the none-efficient storing format of ASCII. Section 6.8 discusses how a novel design can contain every possible height point of the UK, including below sea level within a formatted png texture. Each pixel of the texture represents the height for which is corresponds.

The resolution of the LiDAR maps proves troublesome in respect to processing and also data storage. Each resolution and data point count contained within each map is stated within Table 4.

Resolution	Dimensions	Total points contained	Area Coverage
2m	500^2	250,000	$1km^2$
1m	1000	1,000,000	$1km^2$
50cm	1000^2	1,000,000	$500m^2$
25cm	2000^2	4,000,000	$500m^2$

Table 4 LiDAR data resolutions, dimensions, point count, and area coverage.

What is not shown in Table 4 is that the 50cm and 25cm maps represent the corners of the $1km^2$ maps of 2m and 1m resolutions. To have a map at resolution 50cm, the dimensions will be 2000^2 at total point count of 4,000,000. The 25cm map which covers $1km^2$ will have dimensions at 4000^2 and contain 16,000,000 points.

These maps at even the small resolution and lower point count will still prove troublesome when processing and converting to a 3D model mesh object to be rendered within a game engine. The storage capabilities needed as well will be an issue.

To explain further into the data available, and the amount of data which constitutes this to be big-data, we bring attention to Table 5.

Resolution	Size on Disk including textures	Total size on Disk: DTM	Total size on Disk: DSM	Typical DTM file size	Typical DSM file size
2m	476 GB	234 GB	232 GB	1.81 MB	1.62125 MB
1m	1.23 TB	620 GB	614 GB	5.81 MB	6.585 MB
50cm	663 GB	324 GB	321 GB	6.33 MB	6.1125 MB
25cm	738 GB	364 GB	356 GB	22.6 MB	21.5625 MB

Table 5 Lidar data file size on disk and attributes

Table 5 shows the properties of each resolution of the LiDAR data files; 2m, 1m, 50cm, and 25cm. It shows the total amount of storage on disk for each resolution we have, and the sample size of typical selected maps for the DTM and DSM.

An issue which needs to be stated is area coverage of the LiDAR maps. Unlike the OS mapping, the LiDAR maps are only generated for 1km^2, or 500m^2 tiles, and within flight paths. This creates missing maps in the positions adjacent and diagonally with neighbouring maps. This is problematic with creating a process which combines OS maps, with LiDAR maps, and the multiple resolutions of LiDAR maps we have. A process necessary needs to check the neighbouring maps, and if corresponding data is available, interpolate between the maps, or data points.

4.2.1 LiDAR Conclusion

With the lack of full coverage of LiDAR data for the UK, and where data is available being of multiple resolutions (2m, 1m, 50cm, 25cm), we conclude that addition OS data is needed to create terrain model mesh data for visualisations. Where LiDAR data is not available, OS data will be used. Where LiDAR data is available, but not of high resolution, interpolation techniques will need to be utilised to interpolate to the higher resolutions.

Procedures are also needed to check which neighbouring maps are missing, and load lower resolution maps for combining. Procedures are also needed to check for missing data within the data files, and combine with lower resolution maps which have been interpolated to the higher resolution for combining.

4.3 Total Storage Needs for Complete Coverage

The sections of 4.1 and section 4.2 have shown the size of area coverage for the OS reference scheme, and the number of data-points within the LiDAR data files and the current amount of data we have. This section shows the total amount of storage needed to store the data within its current ASCII format.

Firstly, we state the total coverage of the OS scheme. Due to a majority of the OS mapping scheme covers the water surrounding the UK, and not land, creating a full data set which covers this is unnecessary; see Figure 19 for a depiction.

As stated before, the LiDAR coverage is incomplete for the UK within the resolutions of 2m, 1m, 50cm, and 25cm. We give our findings for the full coverage of the OS mapping scheme as shown in Figure 19. The 50cm and 25cm resolution maps will be spoken as if they have been combined to create an area coverage of 1km^2 as to easily be processed in base 10 format. The reason is that the lower resolution maps of 2m and 1m can be devisable by 10, and this will make it easier to develop for and use with interpolation techniques. To do this, the results of the average DTM and DSM file size for the 50cm and 25cm maps need to be quadrupled to produce the average for a map which of same area coverage of 2m and 1m maps. The results are shown in Table 6.

Map	Average File Size	Partitions	Total File Size
50cm DTM	6.33 MB	4	25.32 MB
50cm DSM	6.1125 MB	4	24.45 MB
25cm DTM	22.6 MB	4	90.4 MB
25cm DSM	21.5625 MB	4	86.25 MB

Table 6 Average size of theoretical 1km squared maps at 50m and 25cm resolution.

The total number of maps from the OS scheme is Layer 1 count * Layer 2 count * Layer 3 count * Layer 4 count. This will be 25 * 25 * 100 * 100. This will produce the value at layer four. We do not need the amount of tiles at the previous layers to be included because we are only calculating the need for 1km^2 tiles.

The result will be shown as 'R' = 6,250,000. To generate the size needed for each resolution of maps, the equation will be R * A, where A is the average for each of the resolutions of the LiDAR at 1km^2 coverage, either the DTM, or DSM. This gives:

Map	Calculation	Size
2m DTM	R * 1.81 MB	11.3125 TB
2m DSM	R * 1.62125 MB	10.1328125 TB
1m DTM	R * 5.81 MB	36.3125 TB
1m DSM	R * 6.585 MB	41.15625 TB
50cm DTM	R * 25.32 MB	158.25 TB
50cm DSM	R * 24.45 MB	152.8125 TB
25cm DTM	R * 90.4 MB	565 TB
25cm DSM	R * 86.25 MB	539.0625 TB

Table 7 Storage needs for full Ordnance Survey Coverage at multiple resolutions.

This gives a total need, which does not include the images of 1514.04 TB of storage. This is the justification to state that this work involves big-data.

Giving the 55 layer 2 tiles we stated, which cover the UK as shown in Figure 22, the storage needs are calculated from the equation 55 * L3 * L4 * A; where A is the average map file size on disk.

Map	Calculation	Size
2m DTM	55 * L3 * L4 * 1.81 MB	0.9955 TB
2m DSM	55 * L3 * L4 * 1.62125 MB	0.8916875 TB
1m DTM	55 * L3 * L4 * 5.81 MB	3.1955 TB
1m DSM	55 * L3 * L4 * 6.585 MB	3.62175 TB
50cm DTM	55 * L3 * L4 * 25.32 MB	13.926 TB
50cm DSM	55 * L3 * L4 * 24.45 MB	13.4475 TB
25cm DTM	55 * L3 * L4 * 90.4 MB	49.72 TB
25cm DSM	55 * L3 * L4 * 86.25 MB	47.4375 TB

Table 8 Reduces file storage for maps within the boundary of the UK Ordnance Survey reference scheme.

The total storage needs shown in Table 8 is 133.23 TB. This amount of data is an issue and will need algorithms designed to process the data.

To deal with the issue of the large amounts of data, we propose an algorithm which can split the UK into manageable 1kilometer squared areas of data. A novel scenegraph structure will be utilised to spatially partition OSM locations to align with the terrain partition. Using this technique, processing only maps which are selected can be achieved using the same scenegraph structure.

To summarise, the issue with data storage needs for LiDAR, and OS, as well as OSM need improving by algorithms, and pre-processors. We contribute to the storage techniques of geospatial data sets, improving, and reducing the need for expensive storage hardware.

4.4 OpenStreetMap

As stated, OSM is a worldwide volunteered mapping platform. It allows anyone with an internet connection to contribute to the geospatial database.

The graph shown in Figure 23 shows the statistics of OSMNodes, OSMWays, and the OSMRelation. The number of OSMNodes within the OSM database is ~= 3.2Billion for the world. Our work will include a subset of this, but the number of nodes per kilometre squared are still large.

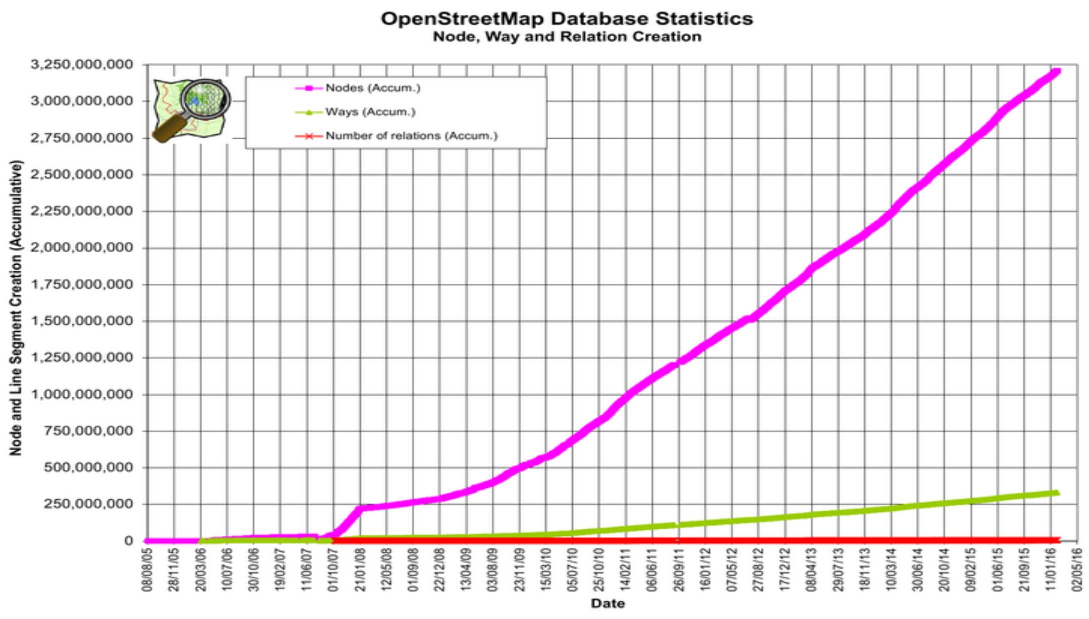


Figure 23 Open Street Map database Statistics. Image obtained from http://wiki.openstreetmap.org/wiki/Stats.

The openness of the platform introduces possible errors. After evaluation of small segments, we have found that errors are included with the downloaded data-sets. To reiterate, OSM contains four main types of data; OSMNode, OSMWay, OSMTag, and OSMRelation. An OSMNode is a single geographical location which can have additional Tag information. An OSMWay is an ordered array of Nodes which represent highways or boundaries; either closed or open. An open OSMWay normally represents a highway or a wall and is stated by the first and last OSMNode not being the same reference. A closed OSMWay normally represents boundaries such as buildings, the first and last OSMNode reference should be the same. A Tag attaches data onto an OSMNode, or an OSMWay and is a key/value pair. A Relationship is a group of Nodes, or Ways, or a mix of both. An example of a Relationship is a building with multiple layers, or with holes within the building. See Figure 24 to view the Liverpool Liver building. Notice the inner cut-outs of the buildings within the figure.

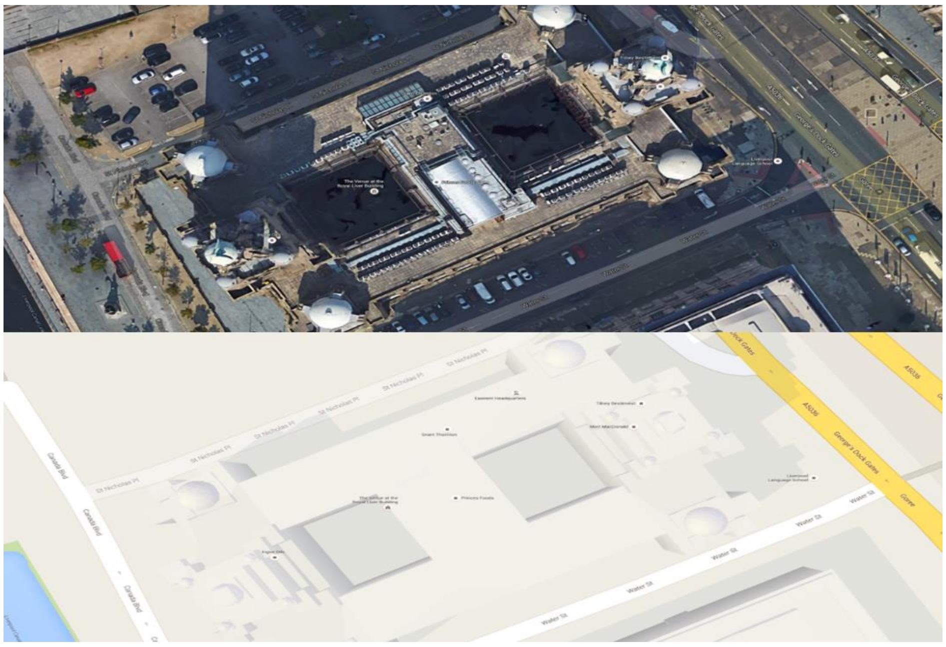


Figure 24 Google Earth and Satellite screen captures. Top has 3D building with images. Bottom is 3D primitive building with no textures. Image obtained from https://www.google.co.uk/maps/@53.405743,-2.9958568,19.25z

The errors which we have identified with the data of OSM are:

- An OSMNode reference IDs are duplicated for multiple Nodes.
- An OSMWay may produce overlapping polygonal boundaries. This issue creates exceptions within the triangulation algorithms when procedurally generating polygonal meshes. Exceptions in this case will crash the program.
- An OSMWay may have additional OSMTags to state that it is a building, but the OSMWay is open; therefore, it cannot be a boundary of a building; this defies usage agreement of OSM.
- An OSMRelation may contain Ways which contain out of order descriptions. For example, a building such as the Liverpool Liver Building within Figure 6 contains cut-outs (the inner boundaries). A relationship which describes this type of building will contain an OSMWay with a Tag with value of 'Outer', and two Ways with Tag value of 'Inner'. The first OSMWay in the array of Ways will be the 'Outer' OSMWay, and the rest will be 'Inner'. Any 'Inner' ways will need a parent of type 'Outer'. Multiple hierarchies of 'Outer' and 'Inner' relationships may occur as well; producing a tree structure. We have witnessed the first OSMWay within the array being of type 'Inner'. Figure 25 shows a representation of this. When processing, checks must be made to remove relationships of this type.
- Tag data can be a great descriptive feature of the database. It can also be a troublesome data type to process due to the ability for volunteers to enter any type of string

representation for the datatype. For example, a colour Tag can be entered in many formats; RGB hexadecimal value and string value.

- Because OSM is a world mapping scheme, it allows many languages to be entered which brings many language parsing issues. We have noticed that the buildings tag information is particularly troublesome. This is accurate with the types of colours attached to a building. The key/value pair of colour is different between English and American English. Users from the UK will use 'Colour', while American English users will use 'Color'. The Key for this needs to be checked.

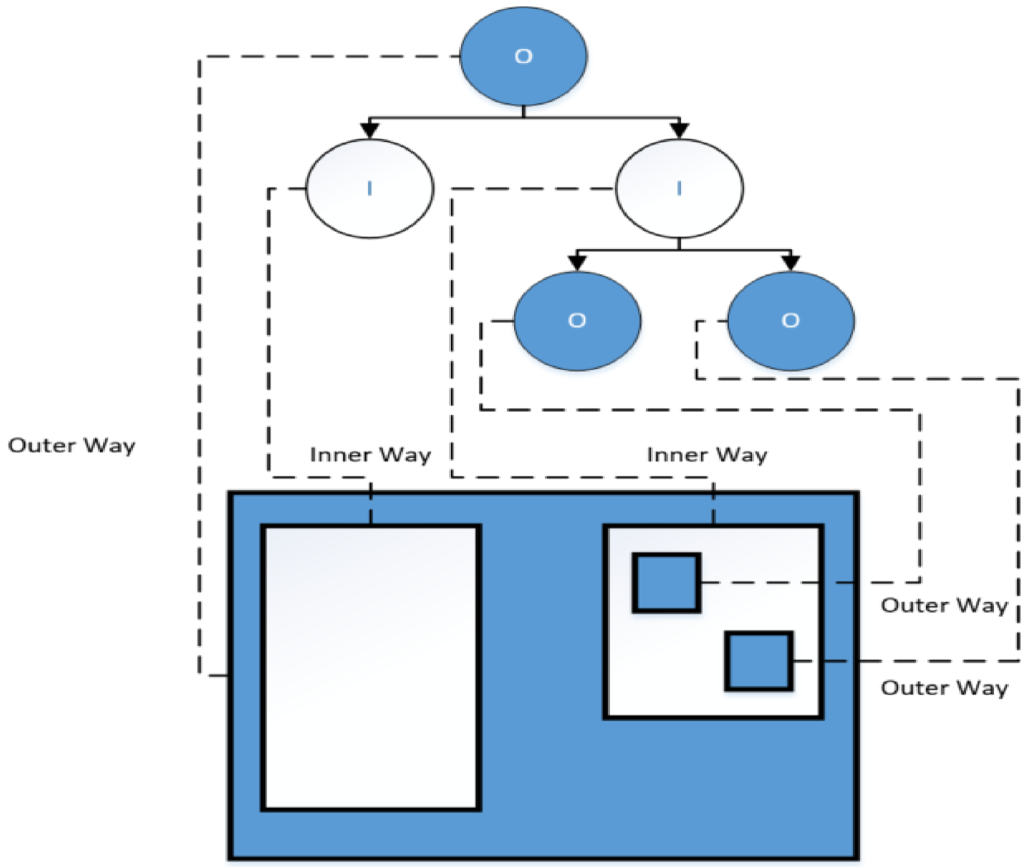


Figure 25 An OSMRelation depicted as a tree structure and visual representation.

The data within the database is stored within the Extensible Mark-up Language (XML) format as stated. Although XML is a regular form to store large amounts of descriptive data, it is unsecure without additional signatures and encryptions, and alternative storage formats are better choices for data storage. One of these formats is binary. XML is a widely used format and already has a wide parsing usage, but binary format will need a custom parsing algorithm. Although binary format will provide improved readability speeds, XML allows widespread usage within other game engines and frameworks.

Due to the 2D nature of the OSM data point locations, procedures need to be put in place to extrude the boundaries to create 3D model mesh objects to view within a virtual environment. Additional procedures are needed to convert from the projection used by OSM to the projection used by OS and LiDAR.

OSM is projected using either Mercator (EPSG 3857) or WGS84 (EPSG 4326) spatial reference system. The Mercator projection is used for tiled web maps such as the OSM website, and that of Google maps. The map area of such maps is a square with x and y coordinates both between -20,037,508.34 on the x axis and 20,037,508.34 on the y axis in meters. As a result, data north of about 85.1° and south of about -85.1° latitude can't be shown and has been cut off. For our research we will only focus on data within the confines of the UK.

The WGS84 projection is used within many Global Positioning Systems (GPS). It uses geo-coordinates between -180° and 180° longitude, and -90° and 90° latitude. Figure 26 shows the mapping of the latitude and longitude upon the globe. It is this projection which is used for converting to alternative projection spaces. This is the projection which will be best suited to project to the X, Y, and Z projection used within a 3D game engine. We will need additional libraries to convert from one projection to another; which proves difficult.

After extensive research, converting from one geospatial projection to another with accurate results is unobtainable. This is due to the curvature of the Earth, specifically the fact that the Earth is non-uniform in nature and 'egg-shaped'. There is currently no software which can accurately convert from the projection used by OSM to X, Y, Z coordinates used by OS and LiDAR data which we have. We aim to monitor the displacement error of the different projections, and create a list of error percentages to improve projection conversion.

The information of the projection system has been researched from OSM projection webpage[36], and a blog webpage[37].

OSM contains limited terrain information stored within its attributes. Additional terrain data is available in part with OSM by SRTM or OpenDEM. OpenDEM is similar to OSM in providing a platform to collect and generate open elevation data.

[36] http://openstreetmapdata.com/info/projections
[37] https://alastaira.wordpress.com/2011/01/23/the-google-maps-bing-maps-spherical-mercator-projection/

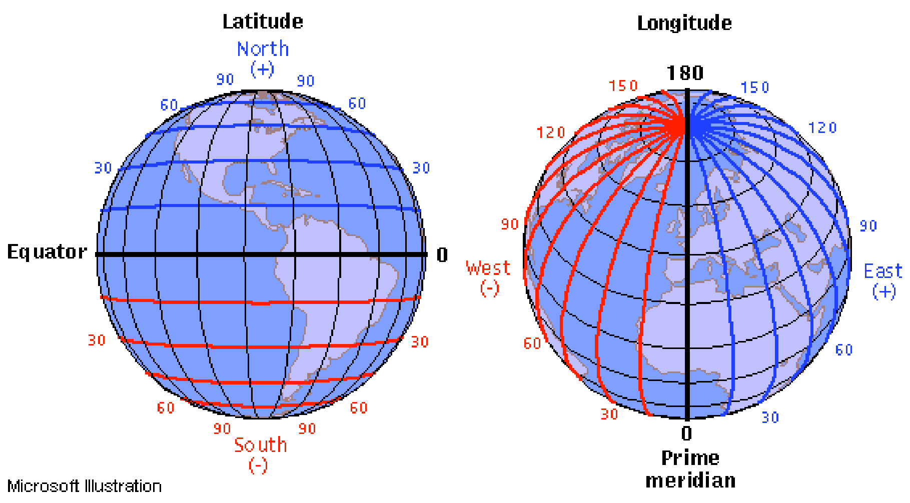


Figure 26 Latitude and Longitude example. Image obtained from http://www.learner.org/jnorth/tm/LongitudeIntro.html

4.5 Conclusion

To conclude this chapter, we iterate the issues of big geospatial data.

The OS format and provider we plan on using will be the OS data provided by the UK government for its similarity with LiDAR data that we have chosen; both use the same coordinate projection of base 10 columns and rows. To generated additional data to fill in the missing data of LiDAR, interpolation techniques need to be utilised. Procedures are to be employed to combine the data sets starting with the low resolutions, interpolate to a higher resolution, and then combine to generate a complete high resolution terrain map. A novel scenegraph structure is needed to partition the terrain maps which total 1514.04 TB of data; or 133.23 TB if creating maps for a subset of the OS projection scheme. The same scenegraph structure is needed to spatially organise the procedurally generated 3D model mesh objects generated from OSM data. Using the two data sets of terrain and 3D model mesh assets from OSM, the conversion between the projection systems can be modified to improve the conversion algorithms which we will utilise from a 3rd party API DotNetCoords[38]

Given the issues with the data held within the OSM geospatial database, additional checks will need to be made either directly on the XML data file or while processing the data within the pipeline.

The next section will discuss the specification needs of the project.

[38] https://www.doogal.co.uk/dotnetcoords.php

5 Specification

Within this section we will discuss the specification needs for the project. The specification states what is needed to achieve the aims and objectives of the project, and to overcome the issues we have identified within the background, literature review, and the big geospatial challenges chapters.

We iterate the aims and objectives of this project, which are as follows;

1. To review and consolidate the knowledge and state of the art on big geospatial data rendering and visualisation.
2. To identify and collect the real world data to generate complex real world 3D environment.
3. To design a framework, and software components to develop a big geospatial data rendering and visualisation system.
4. To create novel data structure to combine several data sources and design new algorithms to process and visualise these data using a 3D game engine.
5. To implement a geospatial data visualisation system.
6. To develop new metrics and benchmarks to evaluate performance of the system of the resulting 3D scenes.

We discuss the specifications from a high-level point of view, then the needs of processing data, the needs for 3D city visualisations, the data structures needed to manage the big geospatial data, and then the algorithms which will be applied to the data structures. Finally, we discuss the use of the Interactive Visualisation Interface (IVI).

5.1 High-Level Specification

This section introduces the high level specification of the framework. The specifications will culminate in the generation of a framework which can process geospatial data into a procedurally generated realistic 3D virtual environment which can be manipulated both spatially and visually by a user.

To generate visualisations for a modern GIS, datasets are available for use, but need additional processes and algorithms to combine the datasets. The combination of datasets is to infer data where data is erroneous or missing. If data is missing, inference algorithms are to be created and used. The use of private and open data will justify the combination as both private and open data has multiple varying errors within the sets. The dataset chosen will guide the development of algorithms and data structures needed for the inference of additional data, as well as the visualisations of real-world assets. As of yet, the combining of this data has not been achieved within industry nor research domains, for the generation of virtual scenes projected and rendered within a game engine visualisation. As stated within the literature review section, similar research is underway but use alternatives to game engines for processing, and visualisations. The generation of 3D assets is the key component, and for this, novel algorithms need to be created. Due to the complexity of asset generation, it is unlikely and inefficient to employ a 3D model artist to generate all possible locations within the UK, or the world; thusly PG algorithms are needed. The algorithms will utilise the data extracted from the datasets, to create procedurally generated 3D assets, or if available, load corresponding user generated content in place of procedurally generating assets.

To allow flexibility with the processing pipeline, from data input, data fusion, data inference, and data generation, a scalable and flexible procedure is needed to allow processes to be removed or added to improve processing speeds, or visualisation outcomes. The flexible nature will allow additional visualisation layers to be added with future iterations, and allow layers to be removed or used within various projects and domains.

Due to the large number of assets and the large number of common rendering techniques employed within computer games, but not yet implemented within GIS frameworks or visualisations of real-world scenes, a flexible rendering pipeline is needed. The flexible rendering pipeline should allow all 2D and 3D model assets to be rendered with any rendering technique with the various lighting calculations. This specifies a unified model object is needed, and the use of a unified rendering pipeline.

To reduce latency with both processing, rendering, and interaction with the virtual scenes, processes must provide techniques to skip processing for the generation of assets. During runtime, algorithms must be employed to organise large scenes to reduce latency during the updating, and rendering of the scene.

The use of the processes within alternative frameworks, or projects is not only needed, but the output of the processes should be flexible enough to use within alternative projects, rendering engines, game engines, or frameworks of the like. A common output schema is needed; binary, XML, GML, etc.

The functional and non-functional requirements are stated next, but first we iterate some use cases for the framework from the perspective of potential stakeholders.

A fire department may wish to view a virtual scene which contains the visualisation of real-world city environments at crisis time. If a building is on fire, the fire department may wish to query a scene, or the objects around a fire to render the likelihood that the queried object will also catch fire. Using this data, the planning of fire engine placement can commence. This same scenario can be utilised in training at non-crisis time.

The police can utilise a virtual real-world scene by overlaying additional data onto processed maps. These data can be visualised by the procedural generation of virtual spheres. The spheres can represent potential incidents, and the size of the spheres represent the severity of the incident. This technique has been implemented for sound level mapping within real-world location within the paper in section 1.4.1, item 4.

The functional and non-functional requirements are stated as follows;

Functional Requirements

- Utilise real-world open and private data. If data is error prone, or missing, then infer and generate data.
- Create a flexible visualisation pipeline.
- Geospatial data parser. This functionality converts between data structures. For example, converting OSM XML format to a runtime class object which can be processed further.
- Geospatial data combiner.
- Geospatial data interpolator.
- Allow scalable and flexible processes and algorithms to combine real world big-data datasets for the reduction of errors and generation of additional data for visualisation.
- Combine big-data data-sets using novel algorithms.
- Procedurally generate 3D assets generated from real-world geospatial data sets.
- Use user generated content in place of procedurally generated assets.
- Data structures to organise a city, country, and a world worth of data.
- Interaction techniques with GPU parameters.
- Advanced rendering techniques.
- Allow users to navigate a scene utilising common 3D camera systems.
- Scene traversal and searching of user generated queries. Traversal and searching of scenes for user satisfied queries should be within real-time, and thusly should not take longer than 2 seconds. This is required by the use case scenarios stated.

Non-Functional Requirements

- Use of commonly used input devices (keyboard, mouse, gamepad) through a framework specific interface.
- 3D model importer.

- 3D model processor. This is stated as non-functional because model assets can be imported through many open APIs. A custom 3D model processor is needed to reduce data storage and apply needed data parameters if the 3D model mesh is imported without them (UV coordinate, tangent and binormals, etc.)

Within the rest of the chapter we will discuss details and issues of each component of the framework. We start by discussing the specification needs for the processing of data.

5.2 Processing Data

We remind the reader that the data sets which we have available are OS terrain data, LiDAR DTM and DSM terrain data, and OSM data.

The issues stated for the LiDAR shows that additional processing is needed to combine the erroneous LiDAR sets with a complete set. The complete set we speak of is OS. A process, or multiple processes are needed to combine the two data sets together. The multiple resolutions of the LiDAR sets mean interpolation procedures are needed to interpolate the low resolution OS set to the high resolution LiDAR sets. Interpolations between resolutions are needed, as well processes to combine terrain sets together, and extract subsets of terrain sets.

In order to determine the most appropriate interpolation function to use, an evaluation of interpolation functions is needed.

The errors which are needed to be checked are; incomplete maps, missing maps, errors such as spikes in height data. Incomplete maps are maps which have data points missing, normally large areas of the map. Missing maps are maps which have not been captured yet; either due to cost, or no need to capture the area (mountains, non-populated areas, seas/lakes). Spikes in height data may represent flocks of birds or mobile artefacts within a scene at capture. Procedures are needed to mathematically model and present these potential errors. Combing OSM may give the additional data needed to query irregular spikes in height data.

The data held within OSM is vast, extensive, and user-generated. The data is also erroneous, lacking, and vastly different depending on the area chosen due to multiple languages, dialects, and understandings of scene being mapped. To utilise the data of OSM, additional checks are needed to ensure the data extracted is suitable for use for asset generation and visualisation. Initial prototypes show that although the OSM API does error check data entered by users, errors are still within the database when downloaded.

The XML file which is used to describe the OSM objects is unreadable and unmanageable without the aid of parsing algorithms. Objects within OSM are referenced with a unique ID, which we have found to be wrong; multiple objects have been found to utilise the same ID, or an object may be entered into the XML file multiple of times. This may lead to additional and unneeded processing and data generation. Another issue is the formatting of input from the users. For this reason, additional checks are needed to remove these errors as to minimise the system failures or duplications of data.

The OSM objects depict assets within the real-world. Parsing the assets into usable data structures are needed. Designing the data structures is also needed. As stated OSM has numerous errors or incomplete data, as the same with the terrain data sets. Data generation techniques are needed to create data where there is no data; through inference or defaulted values created by domain experts i.e. the height of a floor can be estimated through government sectors or private companies[39].

The projections of the terrain data and OSM data are different. An algorithm is needed to convert between the projections; Longitude Latitude to X, Y coordinates as to be used within a game engine.

The information held within OSM used to describe the object lacks information which can be used for analysis and examination. The minimum information to state which OSM boundary outline represents a building is a single Tag object with the key from a key/value pair of 'Building'.

Techniques which analyse the spatial parameters and location of the assets can infer relevant data which can be inputted into the OSM database. This need states that inference algorithms are needed. The inference algorithms will need to be checked by domain experts. For the prototype we will generate our own conclusions on the success of the inference algorithms. If data is not available, nor can it be inferred, then default parameters are needed; again a domain expert should be consulted to determine these default parameters.

To create a visualisation of real-world locations, the generation of 3D model data is needed. Due to the large number of assets, and the need to visualise any location within the UK, or the World, employment of a 3D artist would be expensive, thusly PG processes are needed. The processes will need to convert the information extracted from the datasets into runtime 3D model mesh objects.

Organising the assets within scenes, and within internal processes needs the use of a spatial organisation technique. This is needed to allow data organisation and quick look up of location referenced data.

[39] http://www.ctbuh.org/TallBuildings/HeightStatistics/HeightCalculator/tabid/1007/language/en-US/Default.aspx

The processing of the data needs to output objects into a format which can be used within runtime simulations; a complete processing environment which combines, extrapolates, and generates data from the datasets available to be rendered and interacted with by a user, or a multiple of users.

To improve testing, and remove data which a user may not need, removing unneeded processing, trigger arguments are needed. If only buildings and terrain are needed within the runtime simulation, the removal of processes not used for the creation of buildings will optimise the speed of generation.

We conclude that many algorithms and processes are needed to combine the various geospatial datasets into a single pipeline. Processes are needed to combine the terrain data of OS and LiDAR together, while separate processes are used to convert OSM data assets into model mesh objects utilising PG techniques. To generate accurate and complete data, inference algorithms are needed to improve pre and post model mesh generation which is needed

After the generations of assets, runtime processes are needed to organise and modify assets both spatially, and visually.

Due to the time scale of processing; combining, and interpolations of terrain data, processing the large database of OSM, we specify the ability to allow pre-processing for the creation of runtime assets.

To summarise, the processes needed are;

- Process LiDAR to remove errors/missing data points.
- Processes to combine LiDAR and OS data sets if errors persist within the LiDAR datasets.
- Processes to interpolate through the resolutions of the LiDAR datasets we have.
- Evaluate interpolation functions for accuracy and processing pros and cons.
- Processes to check OSM data parameters and data types, as well as duplicated data.
- Generation of data structures to store and process OSM objects parsed from XML.
- Utilise 3rd party software processes to convert between the projections of LiDAR/OS and OSM.
- Inference algorithms to generate additional data where data is missing.
- PCG processes are needed to create 3D model mesh objects depicting OSM objects; highways, buildings, amenities etc.
- Allow user interaction through an IVI.
- Allow user generated content to be inputted during pre-processing and runtime.

- Allow users to remove unneeded objects at pre-processing time to improve processing speeds.

Utilising the processes stated, a single pipeline can be created to overcome the issues stated within the background, literature review, and big geospatial challenges sections. This data can be used within a number of algorithms and real-time visualisation procedures; creating a large dynamic, and intractable 3D virtual scene which a user can modify within real-time.

5.3 City Visualisation

To visualise modern city's and urban environments, objects stand out as common elements; buildings, highways, amenities, terrain, and objects such as street lamps, post boxes, cars, benches and many others.

To view these objects on a multiple of screens, computers, phones, or TVs, a commercial API is needed to render the objects and manipulate them for interaction needs. A commercial game engine combines a rendering API and the components needed.

A 2D and 3D camera system is needed. The camera system must not only provide an eye within the 3D environments, but must allow the interaction to navigate the scene in a multitude of ways. Common 3D camera systems are built of multi-functional cameras which provide specific interaction capabilities. We will specify the use of the multi-functional cameras. 2D cameras are also needed to view and project overlays within the 3D virtual environment. These are usually Heads-Up-Display (HUD) overlay.

Many commercial GIS platforms provide one of two projections; Orthographic, or Perspective. We specify that a 3D city visualisation must be viewed in both projections.

Before objects are rendered to the screen, a flexible rendering technique is needed. Due to the large amounts of various physical objects needed to create a modern city visualisation, various rendering and lighting techniques, as well as algorithms are needed as to allow the realistic rendering of physically based objects. The rendering techniques needed to render, with realism are; water, glass, plastic, and reflective elements.

The framework should allow all 3D data structures to be rendered with all rendering techniques specified as to minimise the system errors and crashes.

To generated 3D city visualisations, 3D model mesh objects are needed. Creating custom handmade models by a 3D artist is impractical due to many reasons; costs, time, ever changing environments of cities etc. Procedural techniques and algorithms have shown a feasible alternative for the generation of real-world assets. The algorithms stated within the background and literature review sections have been used to generate building models from grammar techniques to a high standard of realism but does not utilise real-world data, while some have created buildings of limited realism and minimal variance in visual representation which is generated purely, or partially of real-world data.

Procedural techniques are needed for the multiple types of assets needed, and the types of objects which are commonly found within real-world locations. Common assets which do not vary are to be generated by a 3D artist; benches, phone boxes, lamp-posts. This is due to their simplicity and constant similar looks. The addition of hand-made models is also needed. These will be used in placement of procedurally generated assets.

To allow the importing of user created model mesh objects, importing, parsing, and processing of the external model datatypes, algorithms which convert the model are needed. This will guarantee the model can be used within the flexible rendering pipeline needed.

To organise all assets spatially within a scene, as well as organise the assets dependent of their type, is needed. Rendering large scenes may not be achievable if the scenes are dense and tightly packed without additional spatial organisation techniques; which many modern urban environments are. Assets should be spatially organised into manageable chunks for improved processing and rendering. A scenegraph structure is commonly used within commercial products and many commercial computer games to spatially organise assets of both static and dynamic nature, from macro to micro elements. The scenegraph structure should provide spatial organisation and categorisation within a single scenegraph structure used for the complete scene. The scenegraph structure can be utilised by other algorithms to infer data, as well as inject data or user input into a single, or a group of nodes.

Rendering objects in a scene for visualisation needs light sources. Within city visualisations, the main light source is the Sun. A global illumination technique is needed as to apply the light source to models, be it sunlight, or spot lights around a virtual environment. To provide flexibility and additional rendering capabilities, assets, specifically the model mesh objects, should be allowed to utilise their own lighting parameters, or utilise the global scene lights. This allows the unique rendering of asset by modifying their spatial or visual parameters, or by modifying their internal reference illumination lights. Modifying the global illumination lights will modify the entire scene and all assets, creating realistic environments.

We conclude a modern GIS visualisation framework which showcases assets within a city requires the use of a commercial rendering API and potential game engine which encapsulates components needed. Before visualisation of city assets, processing of the datasets is needed to convert the data into representations which are apparent within real world city scenes; 3D model mesh objects depicting buildings, highways, amenities etc. The components needed are 2D and 3D camera systems to allow rendering of assets in multiple projections. A scenegraph is needed to position assets within the 3D virtual world, and organise the assets depending on their internal state, and type. To render objects which are imported or procedurally generated, a custom model representation is needed. To utilise and render objects to screen using a large number of rendering and lighting calculations, a flexible rendering algorithm is needed.

We specify that the visualisation of real-world urban environments, light structures are needed. The number of lights corresponds to the rendering technique we specify is needed. We state that forward rendering is needed for this prototype pipeline. It allows flexibility with rendering techniques, and is appropriately chosen for its implementation ease. Deferred rendering may provide a higher number of lights within a visualisation but is unneeded for this prototype.

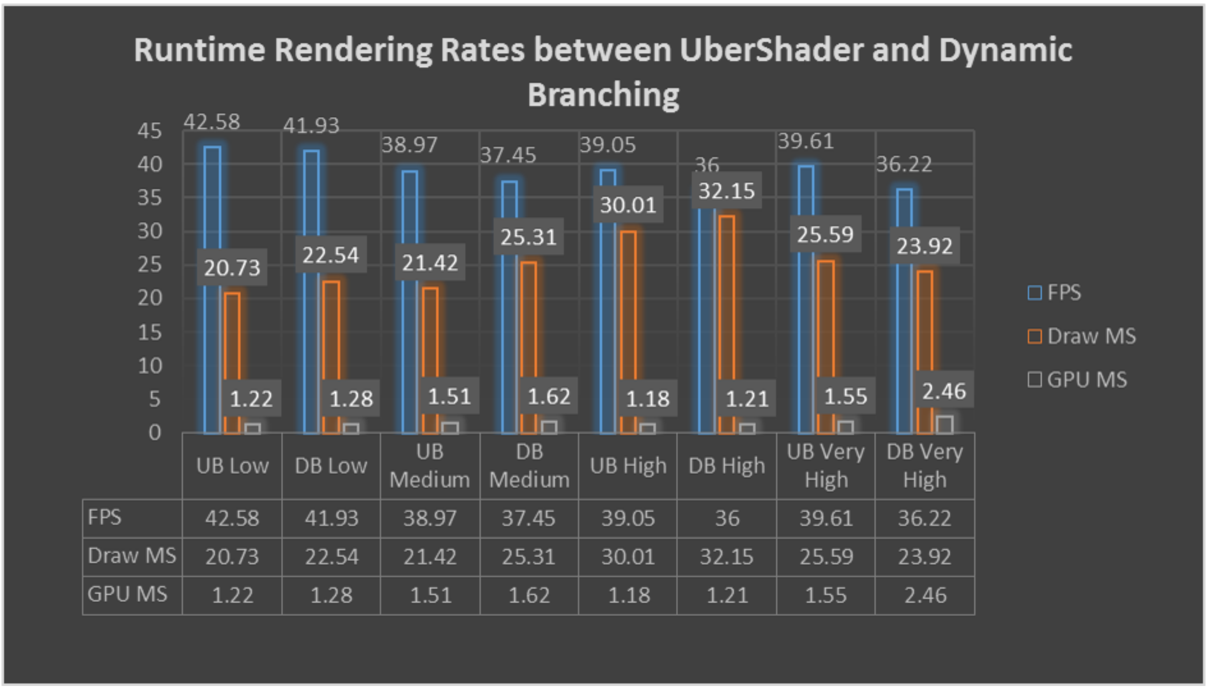
If these specifications are met, large virtual 3D scenes can be visualised, manipulated, and used to present real-world locations inside virtual scenes.

5.3.1 Evaluation of an IASF and multi-branched Shader Effect

The visualisation of large numbers of vertices and polygons needs techniques which do not rely on the high price of dedicated hardware configurations. Scenes to be generated for a modern GIS need high rendering speeds for large, dense scenes. An IASF improves the rendering speeds of large number of vertices over branched shader effects. A multi-branched shader function relies on the Boolean flags of if-statements within the function code of a GPU HLSL shading function. Creating branches is expensive for the GPU and increases rendering speeds due to the increased overheads of each function call. Removing branches from the code base is the roll of the IASF. This increases program duplication, but allows increased rendering speeds. The experiment undertook compared the two techniques. The selected functions of the IASF are chosen CPU side, with each function representing a branch, or series of branches as generated within a shader function with if-statements.

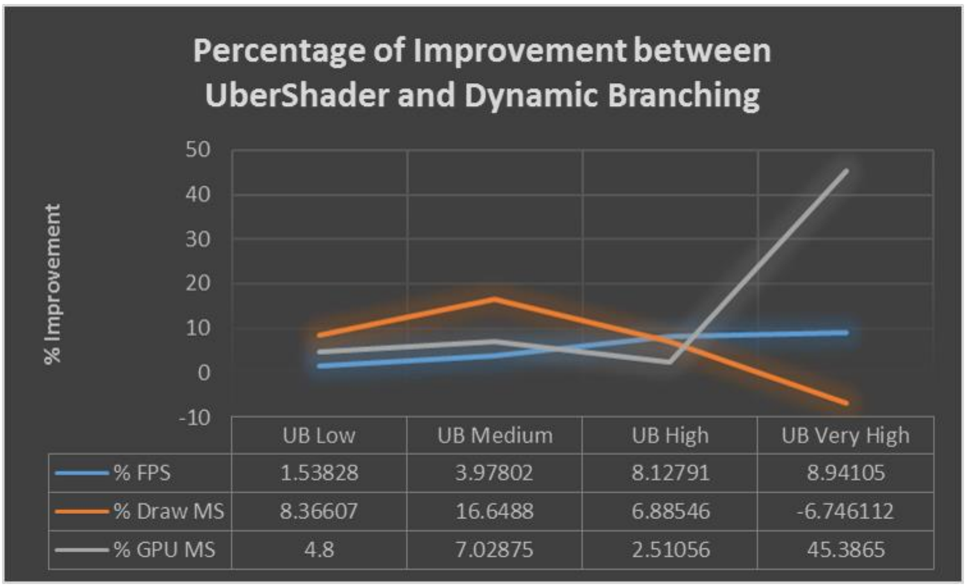
The results shown in Figure 27 show that the use of an IASF to render large buffers of vertex data increases the rendering speed. The experiment to find these results compares a low, medium, high, and very high polygonal scene bring rendered with both the dynamic branching shader, and the IASF. The dynamic branching shader consisted if-statements which chose the lighting calculations; textured, Blinn-Phong, Phong, per-vertex, or per-pixel dependent on the shader technique chosen. The technique would send then pre-defined parameters depicting the needs of the shader. The IASF utilised an indexed array data structure which point to specific functions, which would not pass parameters from technique to function, but simply call separate functions, thusly, removing all if-statements.

Figure *28* shows the percentage gains from using an IASF over the branching shader. Figure *28* shows that the larger number of vertices being processed, the higher the framerate will be compared to that of the dynamic branching shaders which are commonly used. We iterate again that an IASF has not been used within the domain of GIS visualisation and real-time interactions before. With a small number of vertices, the IASF is still beneficial. Figure *28* does show that the draw call is a drawback but the overall framerate is higher, which is the main benefit to utilising the IASF for our work.



	UB Low	DB Low	UB Medium	DB Medium	UB High	DB High	UB Very High	DB Very High
FPS	42.58	41.93	38.97	37.45	39.05	36	39.61	36.22
Draw MS	20.73	22.54	21.42	25.31	30.01	32.15	25.59	23.92
GPU MS	1.22	1.28	1.51	1.62	1.18	1.21	1.55	2.46

Figure 27 Runtime rendering rates between IASF and Dynamic Branching.



	UB Low	UB Medium	UB High	UB Very High
% FPS	1.53828	3.97802	8.12791	8.94105
% Draw MS	8.36607	16.6488	6.88546	-6.746112
% GPU MS	4.8	7.02875	2.51056	45.3865

Figure 28 Percentage of Improvement between IASF and Dynamic Branching.

5.4 Algorithms Needed

We have specified many needs for this project. To complete these specifications and to generate the data needed to visualise a virtual 3D city, algorithms are needed; generated specifically or obtained from open sources. The generation of these algorithms is an essential part for overcoming the problems stated with the datasets available, and the current techniques used by commercial projects and researchers.

To combine the datasets of OS and LiDAR, an algorithm is needed to make the datasets consistent with each other. Interpolation techniques are needed within an algorithm to produce missing data within the terrain data.

An algorithm is needed to extract and infer information from the OSM dataset for the use with PG algorithms. The extraction of OSM data needs domain expertise to create default attributes for OSM assets which have little to no additional data attached. Algorithms to infer information relevant to each asset can be used during extraction, and after generation. For example, the building height can be inferred by defining a constant height for a single floor and multiplying it by the number of floors stated within the description of the building. Algorithms to infer data after generation of assets are to be used to check the spatial relations between assets. Spatial analysis techniques can be employed on 3D assets through the use of commercial physics engines, or collision checks algorithms. Checking if a building is within an OSM boundary, the boundary's type can be used to classify the building; if the building has no type.

Algorithms are needed to procedurally categorise the OSM assets. As stated, the common assets found within modern cities are; buildings, highways, amenities, boundaries, and objects such as benches, post boxes etc. The highways of OSM encompass waterways, and railways. To do this, the algorithm must loop over the OSM dataset, extracting and checking the relevant data. As stated, assets can be categorised, and data can be inferred from other assets within the dataset, such as classification of buildings dependent on their location within the world and if they are within a specific type of boundary, the organisation and categorisation needs to process and generate assets within a specific order. This is to reduce processing and duplications of data.

During classification and processing branching; each branch will generate either a building, highway, amenity etc. additional checks are needed within the algorithm to make sure data has not been duplicated with the OSM dataset.

Once classified, the asset is branched to be procedurally generated. To allow hand-made models, an algorithm is needed to check the hard disk whether a model exists for the currently processed asset. If a model does exist, the algorithm should load and process the model to use it in place of the procedurally generating asset. This should be done for all objects. With OSM unique identifying assets with their own ID number, the lookup of models can be procedurally checked within the algorithm.

If a model object is not referenced on disk to be used instead of procedural generating assets, an algorithm is needed to check the type of the object, and if it is a common type, such as a residential building, a general purpose model should be used. This removes the limitation of primitive procedural assets needed custom complex algorithms.

If no model object is to be used, procedural algorithms are needed to create the 3D model mesh objects. Procedural algorithms are needed for each asset to be generated.

The algorithm to generate boundary model mesh objects will need to utilise a triangulation algorithm to generate the model mesh object from an arbitrary array of points.

The algorithm for the PG of buildings needs to account for a building which is generated by a multiple of parts. The roof tops of buildings will utilise the triangulation algorithms used to create polygonal model mesh objects.

The algorithm for the PG of highways, waterways, and railways will need to utilise interpolation techniques to interpolate between the data points. We reiterate the highway structures held on OSM are represented by a limited number of points when visualised create angled lines. Figure 29 represents the procedure needed.

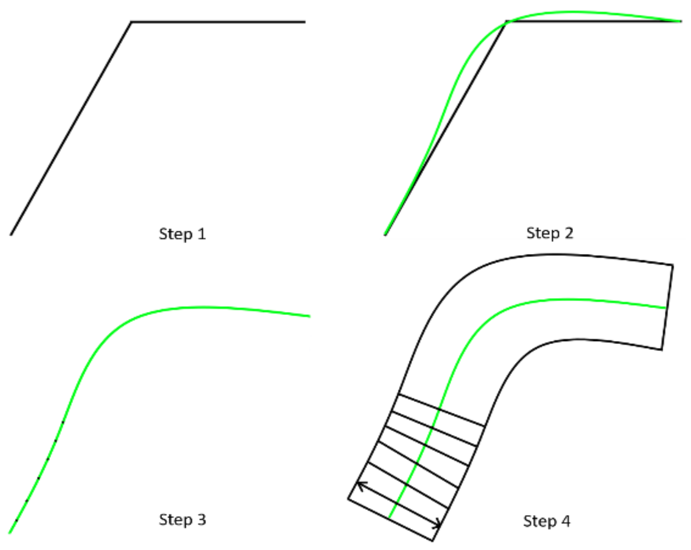


Figure 29 Highway mesh generation. Step 1 is OSMWay representation. Step 2 is the result of the interpolated points. Step 3 shows the points generated from the interpolation. Step 4 generates the perpendicular sides of the highway.

During the runtime simulation, rendering algorithms need to provide optimisations to categorised assets depending on their rendering needs; translucent, opaque, and others. The algorithms must also allow the changing and procedural setting of rendering techniques depending on the needs of the model mesh object. For example, to render reflective water, the model mesh asset must be set to utilise reflective rendering techniques. This type of rendering technique is process intensive, so an algorithm is needed to monitor runtime simulation details such frame rate and procedurally set the rendering technique to a lower process intensive technique to improve rendering rates to allow for real-time visualisation.

For a group of nodes within the scenegraph structure, or a group of model object to be modified, search techniques must be employed to allow the procedural querying of assets within a scene. The user will create a query, which an algorithm will interpret and search a scene for objects which satisfy the query. This leads to multiple needs from the algorithm; search techniques to be created which work with the scenegraph layout, the PG of queries, and the need to do something with the assets satisfy the user generated query.

Once an asset or assets are found from the query, controllers will be employed to modify specific properties of the spatial parameters, or the visual parameters. From our background and literature review, the animation of properties of assets within a virtual scene depicting geospatial data has not been used as of yet. An algorithm which exposes all properties of the spatial parameters, and visual parameters is needed to allow the animation, interpolation, and modification of the properties. The algorithm will allow users to encode their own datasets and meanings onto a cities worth of assets. For example, querying for buildings which are over 20 meters, and not fire-proof will search the scenegraph for building which satisfy this query, and a user can state to render the assets in red, or pulsate the assets scaling, bringing the attention of the user to the returned assets.

To animate objects, controller objects should be employed to update the parameters of the asset which the controller is attached to. This will improve the processing of assets; if an asset does not have a controller attached, then it does not need to be processed.

Various optimisation algorithms are needed to improve the rendering and batch processing of assets. For example, the scenegraph structure can be searched for assets which have a controller, and put these assets within a referenced array used to linearly update the assets, instead of recursively traversing a scenegraph checking if assets need updating.

We conclude that generating the algorithms stated will remove or partially remove the issues surrounding modern GIS visualisations and allow for large, accurate rendering and interactions with procedurally generated 3D city scenes. We have stated the need for algorithms to parse, process and combine terrain datasets to be utilised with the inference of data for the assets which are procedurally extracted from OSM datasets. The inference of data is also programmatically checked within the processing of OSM data creating further accurate asset objects. The assets are then procedurally generated to create 3D model mesh objects. If a model is already available and suitable to be used in place of the procedurally generate asset, then it will be processed and referenced accordingly. Once PG of assets is completed, they are to be used within a post-processing algorithm to infer further information. During runtime visualisations, algorithms are utilised for animating and modifying assets within the scene, allowing the flexible rendering techniques to be applied to single assets, or complete scenes of assets. These techniques are to be encoded and dispersed through the scenegraph structure, allowing a flexible, single pipeline process for a modern GIS visualisation framework. Dispersing data through the scenegraph from the top to the bottom can change a scene visually within a single recursive function call.

Next we discuss the requirements for the User Interface.

5.5 Interactive Visualisation Interface Requirements

The requirements of the IVI system will provide interaction with the algorithms and runtime simulations, modifying the spatial and visual properties of assets within the scene, and allow user defined queries to be applied to scenes. This will improve overall functionality of the framework, searching, and provide flexibility with the visualisations of scenes. As well as visualisation flexibility, the interaction of the IASF parameters is achieved, extending the functionality and benefits that the IASF brings to large scale scene visualisation. The use of the IVI will also allow inspection of the PCG, and specification of the procedurally generated models which need to be visualised first, and then inspected.

The IVI system must allow multiple functionalities and interactions. We have stated a scenegraph structure is needed to spatially organise and classify assets within the scene, the IVI system must allow interactions with that scenegraph structure. The creation and deletion of nodes, and the spatial modification will allow adaptation of the scenegraph to alternative needs.

The IVI system must also allow the searching of the scenegraph for a single or group of nodes selected by the user's procedural queries; querying the various data structures or information attached to a node. Allowing queryable functionality on a scene, or an individual asset, the framework can be utilised within a number of domains, regardless of the background knowledge of the user; the fire brigade can utilise the framework to search for none fire-proof buildings, while the police can use it to search for highways of specific width to navigate large vehicles through a city.

As stated, once an asset or a group of assets has been chosen, the generation of control units which modify the properties of assets such as spatial parameters, and/or the visual parameters are needed. Allowing the generation through an IVI system, where the user can select the parameters to change, the animations of that parameter, and the values to modify the property too is needed.

To allow modification of the global lighting system, the attributes of the lights must be presented to the user to allow the changing of properties of lights such as; colour, position, direction etc. Assets within the environment may have their own internal lighting parameters, which must also be presented and modified if needed.

The modification of individual assets concludes that an IVI system depicting all asset parameters, attached objects, and interactions is needed.

For development, and further analysis techniques, debugging IVI systems are needed for the internal components of the system; cameras, assets, global lighting, timers for rendering, GPU processes, frames per second and others. This will provide insights into issues within the system. Also they can be used as teaching aids for new users of the framework.

To allow the selection of areas with the UK, the system needs to complete multiple tasks to enable the user to select a 1km^2 area tile to load from the pre-processed data. The selection of the tile will navigate the user through the OS reference scheme from 500km^2, 100km^2, 10km^2, 1km^2 area tiles. At each stage of selection, the hard disk will be scanned for areas which have been created, and the tiles which are available on hard disk will be highlighted for the choosing by the user. Once a 1km^2 tile is chosen, the middle out loading algorithm will load that 1km^2 area of assets for viewing and interaction by the user. We issue this will allow the user to know which areas are already processes, and guide their selection of areas.

The selection of assets/objects within a scene, a user must be able to query the objects on the properties of the object. For example, a user may wish to find a building which is fire-proof and over 20m's in height. A system must allow the querying of said properties of an objects, in any order they wish. The query should state that properties return true. If any of the query parameters are false, the objects will not be returned. An example of the query may be similar to using unary operator to query the properties of an object;

building.height > 20 && building.fireproof == true

The query will check if the building has a height about 20, and the fireproof property is equal to true. This is a common querying definition. The framework must also allow a user to type the query in, and procedurally convert the text to a valid query.

In conclusion, the IVI system must present information of the assets in the 3D virtual scene, and the nodes within the scenegraph structure. If a component interacted with either a node or nodes of the scenegraph, or the model mesh, then component properties must be presented to the user, and modifications must be allowed. The same is said for the node of the scenegraph and the model mesh object. Users of the framework can then manipulate the scene in a manner which suits them, and allows the encoding of their own data to be rendered within the scene by the PG of custom controllers.

The use of the IVI will provide functionality not available with the alternative GISs; Google maps, ArcGIS, SAVE etc. The IVI also allows real-time interactions with scenes, allowing the modification of properties on a frame-to-frame basis.

5.6 Software Development Life Cycle

This section discusses the need for a development lifecycle model for the project. Balaji [75] gives a discussion between the Waterfall model versus the V-Model versus the Agile model. The author states that the project itself determines the life cycle needed.

- The Agile [76], [77] lifecycle should be chosen if changes are frequent and the need to deliver small subsets of a larger framework.
- The Waterfall model [78] should be chosen if the requirements are fully known before development begins.

- The V-Model should be chosen for larger project which requirements may change after development, testing, and delivery of each phase of a project.

We state that the Agile model would be a sufficient model for this project. This is due to the need to rapidly prototype sections of the framework, and to further prototype the combination and interaction between multiple prototype for the generation of a working prototype framework.

5.7 Conclusion

We have specified the needs of the project for the creation of a single pipeline to combine, extrapolate, infer, and generate data from available geospatial datasets. We have presented the processes needed to combine the datasets. We have also presented the novel spatial data structure and algorithms needed to combine and organise the data during visualisation and rendering processing. The algorithms are responsible for the generation of additional data and assets, specifically the generation of 3D model mesh assets used for visualisations. To interact with the runtime simulation, an IVI system is needed to modify assets and object within the scene. The IVI system is also needed to present data to the user, as well as data to the developers.

Table 9 shows the specification requirements conclusion.

Object	Description
Agile development lifecycle	The Agile development lifecycle is chosen due to its benefits of design patterns it brings to projects with known unforeseen issues.
Geospatial data parser	The geospatial data parser is needed to load various geospatial data sets into runtime objects for data analysis.
Geospatial data combiner	The need to combine various data sets is needed to fill missing data, and amend errors.
Geospatial data interpolator	Interpolation algorithms are needed due to the characteristics of the terrain maps, and model mesh generation techniques. LiDAR maps have missing data which can be generated by interpolating between know data height points.
Geospatial data inference algorithms	Inference algorithms are needed due to the characteristics of the datasets at hand. OSM has data which can infer alternative data; the height of a building can be calculated by multiplying the number of floors by a default floor height value.

Data structures to organise a countries worth of data	Due to the size of the area we wish to tackle, a single kilometre, a country, and potentially a global mapping schema, the organisation of assets is needed. The organisation of location, and also the data type is needed to be encoded into a single data structure.
3D model importer	To limit issues with rendering techniques, all model mesh objects needs to contain the same data needed for rendering (Normal, UV coordinates, Tangent, BiNormals etc.). This removes issues at runtime to check if a model can be rendered using a chosen rendering technique. Default data should be applied if data is missing while importing.
3D model processor	The processing of procedural model mesh objects created from geospatial data is needed. The output will be the same as a user generated 3D model assets, containing the same data requirements for rendering.
Procedural algorithms for model mesh generation	Due to the differences with the geospatial data; terrain, buildings, boundaries, etc. procedural generation algorithms are needed for each type of object. The algorithms are determines by the attributes of the imported geospatial data.
Advanced rendering techniques	Advanced rendering techniques corrispong to the use of an Array Indexed Shader Function structure, but also the shader effects applied to the model objects; normal mapped, textured, refractions, hemispherical ambient lighting and many others.
Allow user generated content importing	The inclusion of user generated assets (3D model meshes) is needed to improve scene renderings.
Interaction techniques with GPU parameters	The use of materials used by models is needed. These materials are exposed to the user to allow the interaction with GPU settings and material parameters.

Table 9 Specification requirements conclusion.

Given the specifications listed in Table 9, the high-level overview of the framework is shown in Figure 30. The pre-processors contribute to the generation of data which is imported by pre-processor and runtime libraries into the runtime simulation. The data is stored within separate data storage devices and combined to procedure virtual scenes organised by a runtime scenegraph structure.

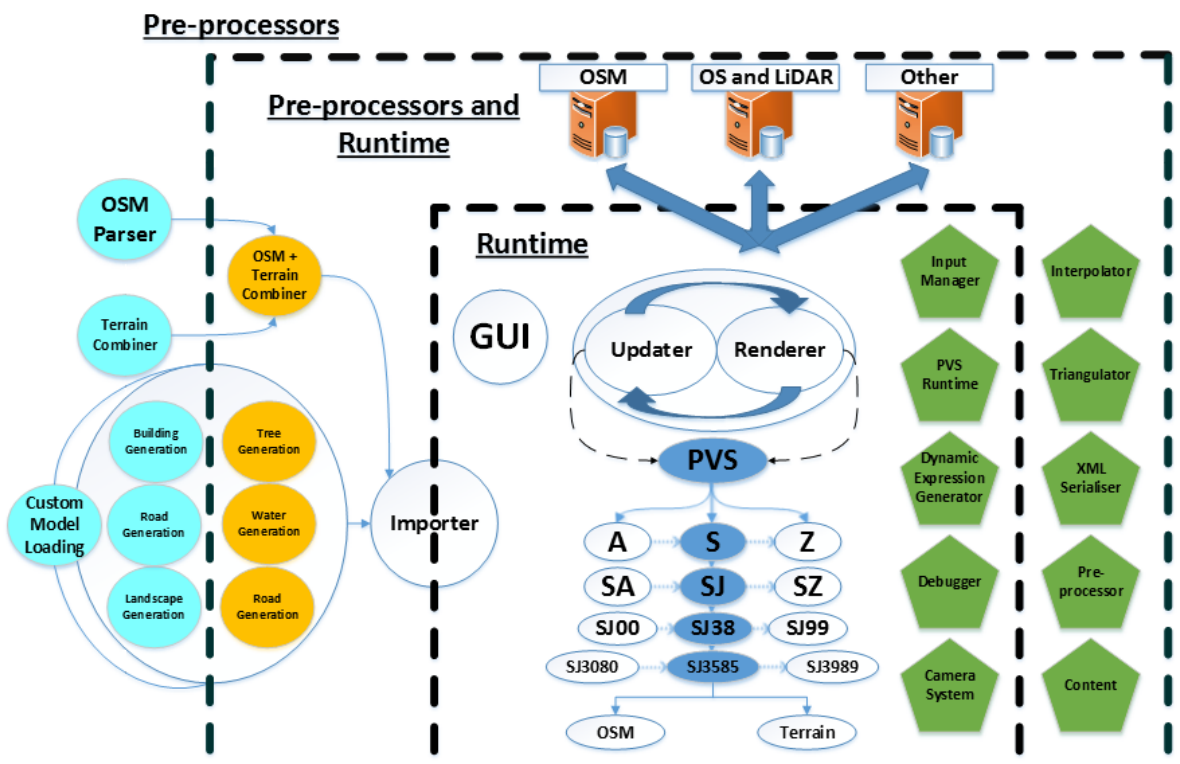


Figure 30 High level framework overview

Designs of the novel PVS framework and related data-structure and algorithms to overcome the issues discussed will be presented in the next chapter.

6 Design

Within this section we discuss the high-level overview of the design of the PVS framework and design of components needed as identified within the specification.

Objects created for the framework, will be named with the subscript of *PVS*. This is to remove confusion between OSMNode, PVSNode, and node. An OSMNode is a single location within the OSM database. The PVSNode is a node within the Scenegraph used within the PVS framework. A node is used for discussion of scenegraphs, and tree structures.

6.1 High-level Overview

Figure 31 shows the high-level overview of the framework. Each block represents a library. Within the libraries, classes are stored. The class objects, and the connections are coloured differently as to easily depict the interactions and connections between them.

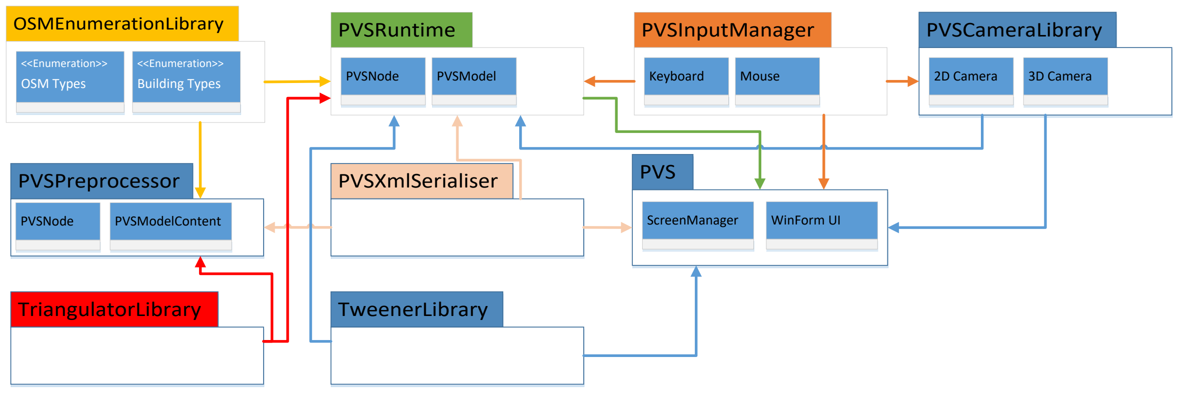


Figure 31 High Level Overview of PVS Libraries

Figure 30 depicts the high-level over-view of the framework with further visualisations of the connections between libraries, and data structures, including input data. The framework is split between a pre-processor, and runtime simulation.

Pre-processes are run before user simulation. The pre-processes are to be used by the user to generate the input for the runtime simulation. The processes are designed to be run in succession or individually. The pre-processes contain parsers, importers, and processors to load the datasets, convert user generated model mesh objects, and it contains the PG algorithms for the PG of data and model mesh assets. Pre-processing data has many benefits. When data is modified daily, as with OSM, pre-processing can be triggered to generate up to date runtime data for accurate simulations. This leads to the need for a pre-processor pipeline, which can export assets for use within a runtime simulation.

Figure 30 shows algorithms, processes, and libraries which are categories to be used within both the pre-processor and the runtime simulation. These consist of PCG algorithms, utility classes such as the OS references, as well as the flexible and modular libraries. The libraries consist of; interpolation functions, triangulations algorithms, XML serialisation, access to the pre-processor components, and the content either procedurally generated or user created (textures, models, animations).

The Runtime processes shown in Figure 30 are algorithms and processes which are to be utilised within the runtime simulation. This consists of flexible custom rendering algorithms utilised within the novel scenegraph structure for rendering a large number of various model mesh objects. The IVI is utilised within the runtime simulation to allow interaction with the scene from the user. To allow users to view scenes, the modular 3D camera system is only instantiated within the runtime simulation. Combined with the modular 3D cameras are the input libraries to allow keyboard, mouse, and gamepad input. The dynamic querying library allows a user's query to be applied to a scene, searching assets which match the query criteria.

The scenegraph and the PVSNode within a scenegraph are referenced throughout the design, and are a key component to the framework. Thusly, we highlight first the design of the PVSNode within the scenegraph, and the interconnections that the PVSNode will have with other components of the framework such as; spatial controllers, material controller, tag information, 3D model meshes and model mesh parts, model materials, IASF, global light manager, IVI, data importers, parsers, processors, writers and readers, 2D and 3D camera system, and user input API.

The design within Figure 32 shows the PVSNode within the scenegraph. It shows an inheritance from the PVSSpatial to the PVSNode which has an affiliation with the Search Helper which will be a collection of search techniques and algorithms. In short, the PVSSpatial contains the model mesh object, and spatial parameters (translation, rotation, scale). The PVSNode extends the functionality to contain an array of child PVSSpatial objects. Within a PVSNode of the scenegraph, a PVSTag is to be attached, as well as possible spatial controllers, and a possible model object. We state that a spatial controller and model mesh object are attachments, thusly, a node without these attributes can still function. Pictured to the top right of Figure 32 is the affiliation to the global light manager. The light manager has restricted effect on the light conditions of models attached to PVSSpatial. The OS Ref object is a reference to the OS tile name lookup-table which will be used to look up neighbour nodes of the scenegraph used within loading and bounding volume culling techniques. The model object contains model parts; each have a material for storing parameters used by the GPU for rendering. The parameters are passed to the GPU using the Material Helper. The Material

Helper optimises parameters, group objects together CPU side, then passes the parameters to the effect object on the GPU.

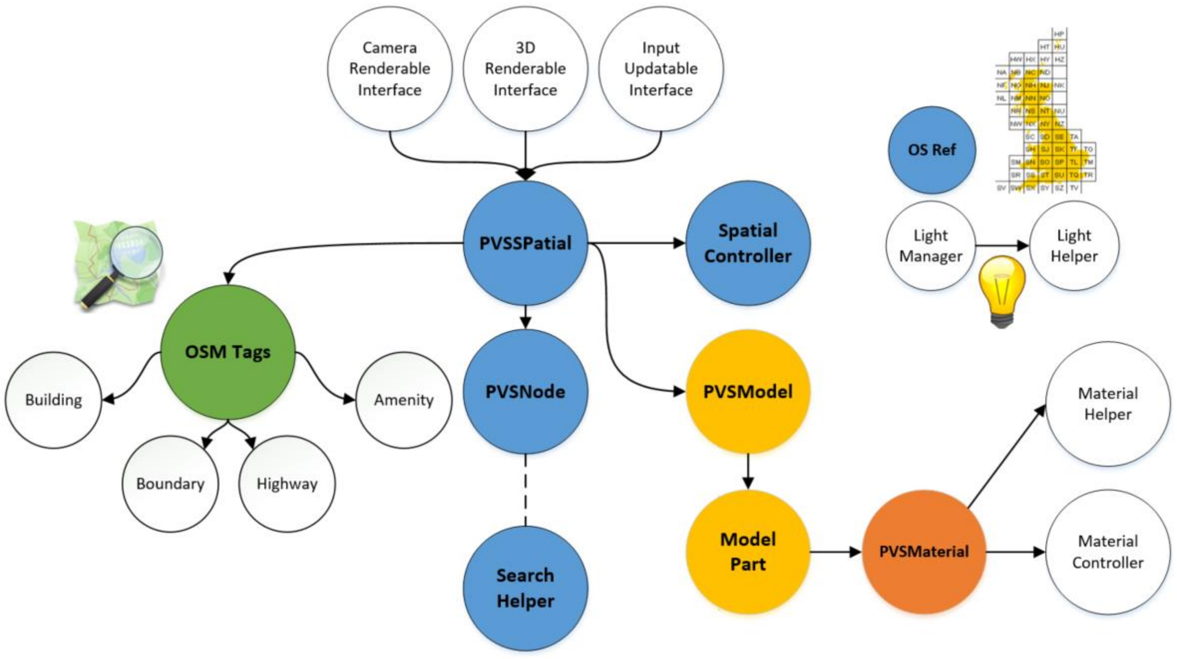


Figure 32 PVSNode scenegraph node.

We have stated in the specification stage that a flexible rendering pipeline is needed. A technique which can encapsulate all shading techniques needed to render common materials found within real-world urban environments is a requirement. We state the use of an IASF design will satisfy the specified needs. The common name for this technique is an UberShader (US). The IASF will contain all vertex shader and pixel shader functions, each indexed into an array lookup table. This allows an object to be rendered by modifying a single index value, instead of creating and referencing a complete effect object; improving flexibility, scalability, and optimising data usage.

There are two design decisions when utilising an IASF which encapsulates all rendering options within a single effect file. The first is; the need for generation of complex algorithms to check the model mesh data and assign an appropriate rendering technique, and also check that a model object is not forcing a render technique inappropriate for itself which will cause system failure. The second is; check if all model mesh objects contain default parameters needed for rendering, using any of the rendering techniques within the effect file. The latter is the better choice due to its simplicity, although the duplication of shader code may incur maintenance issues, but it will allow for many types of model mesh objects to be rendered. It also allows for future rendering techniques to be quickly implemented.

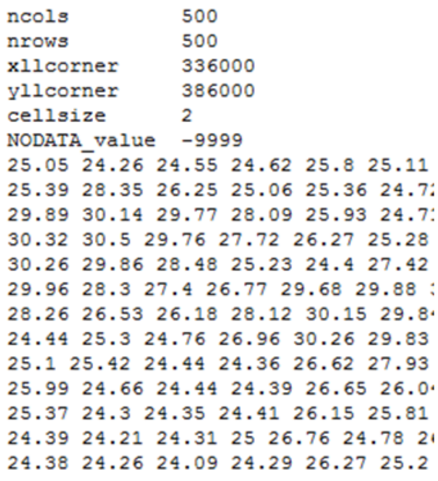
Developing a framework depicted in Figure 30 and a scenegraph node structure shown in Figure 32, large scene visualisations of cities can be achieved. We discuss further the designs of the components.

6.2 Input Data

We have described the input data for the framework in the sections of Background, Literature Review, and Big Geospatial Data Challenges. The input data consists of OS, LiDAR, which represent the terrain data needed to create terrain model meshes. OSM is used for the creation of buildings and individual assets within visualised scenes. User generated content is also imported, and accompanying text based schemas which apply translations, rotations, and scaling to imported user generated 3D models.

The terrain data of OS and LiDAR consists of ASCII based text files which contain Meta data. OS and LiDAR Meta data differ slightly, but both contain map resolution, width and height. LiDAR contains missing data values of -9999 which can be exploited within algorithm to check for errors.

The ASCII files consists of rows and columns of ASCII based numbers separated by a space character. Figure 33 shows the ASCII text format and corresponding terrain representation. The red dots represent missing data values.



```
ncols         500
nrows         500
xllcorner     336000
yllcorner     386000
cellsize      2
NODATA_value  -9999
25.05 24.26 24.55 24.62 25.8 25.11
25.39 28.35 26.25 25.06 25.36 24.7
29.89 30.14 29.77 28.09 25.93 24.7
30.32 30.5 29.76 27.72 26.27 25.28
30.26 29.86 28.48 25.23 24.4 27.42
29.96 28.3 27.4 26.77 29.68 29.88
28.26 26.53 26.18 28.12 30.15 29.8
24.44 25.3 24.76 26.96 30.26 29.83
25.1 25.42 24.44 24.36 26.62 27.93
25.99 24.66 24.44 24.39 26.65 26.0
25.37 24.3 24.35 24.41 26.15 25.81
24.39 24.21 24.31 25 26.76 24.78 2
24.38 24.26 24.09 24.29 26.27 25.2
```

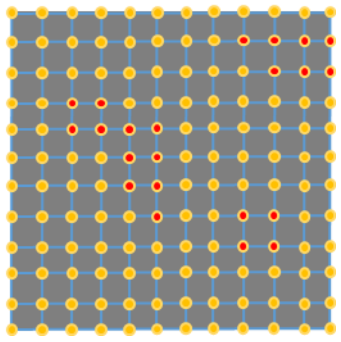


Figure 33 LiDAR data input. Left is ASCII text file. Right is terrain map representation with red points as missing data values.

The OSM data is in XML format which needs to be serialised into a runtime class object. The OSM data file is extracted from the OSM web portal[40] which contains the information within the selected section. The XML consists of; OSMNode, OSMWay, OSMRelation, and OSMTag. As stated the XML is serialised to a runtime object which will contain arrays of OSMNodes, OSMWays, OSMRelation, and

[40] http://www.openstreetmap.org/

OSMTags. This data is then used within algorithms to infer or create data. The locations of the nodes are referenced with Latitude and Longitude, which need to be converted to X and Y coordinates.

The user generated 3D model assets are of FBX format[41] and parsed into a custom format. The files consist of 3D model mesh vertices, diffuse textures, normal map textures and specular map textures, as well as a material object and possibly an effect object. The effect of the FBX model needs to be analysed and the appropriate effect functions need to be applied within the parsing of the model. This removes the need to store various effect objects, using only the IASF. If textures are not applied, then default textures are to be applied during parsing. This allows the 3D models to be rendered using common rendering and lighting calculations.

With user generated content, the translation, rotation, and scaling may differ to that of OSM bounds, thusly additional user generated content is needed. This is in the form of a text file with a schema of;

<**OSM ID**> <**File Name**> <**X, Y, Z Translation**> <**X, Y, Z Rotation**> <**X, Y, Z Scale**>

Using this schema, data can be stored on a per-model basis. This data can then be used during content creation. Figure 34 shows a user generated 3D model mesh of the Royal Liver Building. This is translated to the location of 0, 0, 0 due to simplicity of generation, but using the scheme, it can be translated, orientated, and scaled accordingly without need to modify the model within an external 3D modelling software.

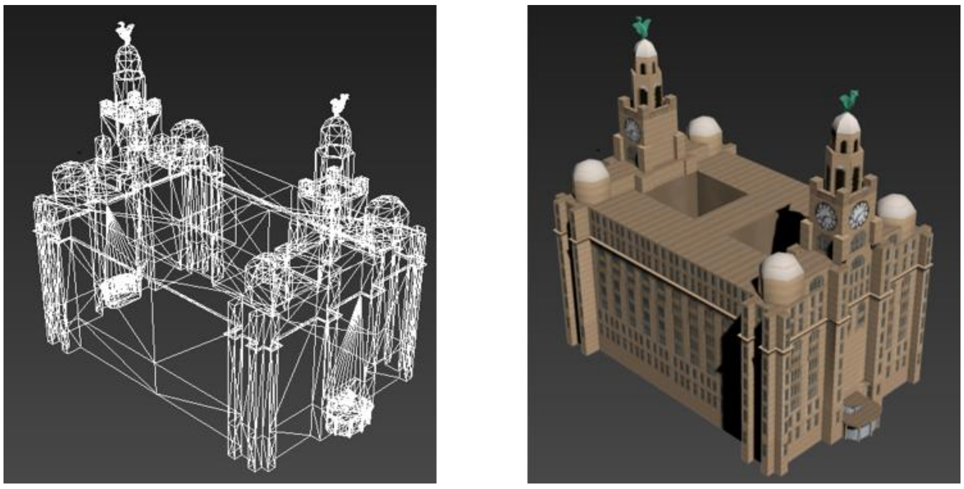


Figure 34 FBX user generated model input, wireframe and model.

[41] http://www.autodesk.com/products/fbx/overview

User generated animations of spatial and material parameters are input at runtime. The user generated controllers extend the functionality of the visualised scene; applying visualisations to assets within a scene.

OSM data is represented and stored as binary and XML. Utilising XML the generation of custom parsers are achievable. The snippet of XML shown in Figure 35 represents the Royal Liver Building as depicted in Figure 34 and Figure 36.

Objects within OSM are represented using single latitude and longitude locations stored in OSMNodes. Grouping OSMNodes create OSMWays which represent lines, or boundaries. Grouping lines or boundaries are achieved by OSMRelation. To describe the OSM object, OSMTags are stored within the OSM objects. OSMTags are a <*Key, Value*>. It is from the OSMTag objects which assets can be queried and classified.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.5.8 (1016 thorn-01.openstreetmap.org)" copyright="Open-
StreetMap and contributors" attribution="http://www.openstreetmap.org/copyright" li-
cense="http://opendatacommons.org/licenses/odbl/1-0/">
 <bounds minlat="53.4053900" minlon="-2.9963400" maxlat="53.4054900" maxlon="-2.9961900"/>
 <node id="1676134606" visible="true" version="1" changeset="10992179" timestamp="2012-03-
15T19:58:17Z" user="UniEagle" uid="61216" lat="53.4061925" lon="-2.9954691"/>
 ...
 <node id="1676134609" visible="true" version="1" changeset="10992179" timestamp="2012-03-
15T19:58:17Z" user="UniEagle" uid="61216" lat="53.4062037" lon="-2.9955510"/>
 <node id="267535473" visible="true" version="5" changeset="10992179" timestamp="2012-03-
15T19:58:45Z" user="UniEagle" uid="61216" lat="53.4062223" lon="-2.9954991"/>
 <way id="24611033" visible="true" version="10" changeset="32552498" timestamp="2015-07-
10T20:18:30Z" user="Med" uid="88164">
  <nd ref="267535473"/>
  <nd ref="1676134606"/>
  ...
  <nd ref="1676134609"/>
  <nd ref="267535473"/>
  <tag k="building" v="yes"/>
  <tag k="building:colour" v="#786B56"/>
  <tag k="building:material" v="stone"/>
  <tag k="building:part" v="yes"/>
  <tag k="height" v="51.60"/>
  <tag k="layer" v="1"/>
  <tag k="name" v="Royal Liver Building"/>
  <tag k="tourism" v="attraction"/>
  <tag k="wikipedia" v="en:Royal Liver Building"/>
 </way>
 <relation id="2084603" visible="true" version="1" changeset="10992179" timestamp="2012-03-
15T19:58:32Z" user="UniEagle" uid="61216">
  <member type="way" ref="24611033" role="outer"/>
  <member type="way" ref="155196231" role="inner"/>
  <member type="way" ref="155196230" role="inner"/>
  <tag k="type" v="multipolygon"/>
 </relation>
</osm>
```

Figure 35 OSM XML file example.

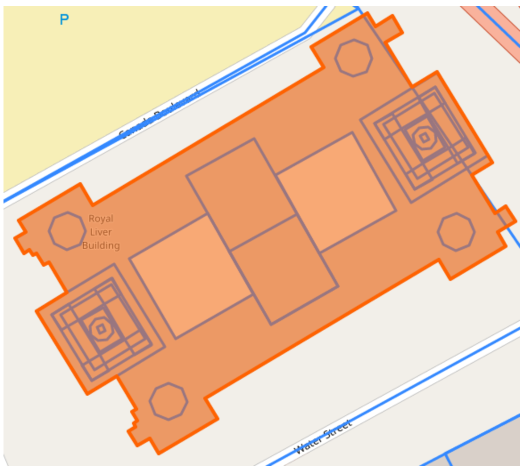


Figure 36 Royal Liver Building representation on OSM.org

6.3 Pre-processing Data

The primary goal of the pre-processors' is to generate output files for use within the runtime framework and within other frameworks, game engines, and simulations. Each output file will represent a PVS Node from the scenegraph which will be loaded at runtime.

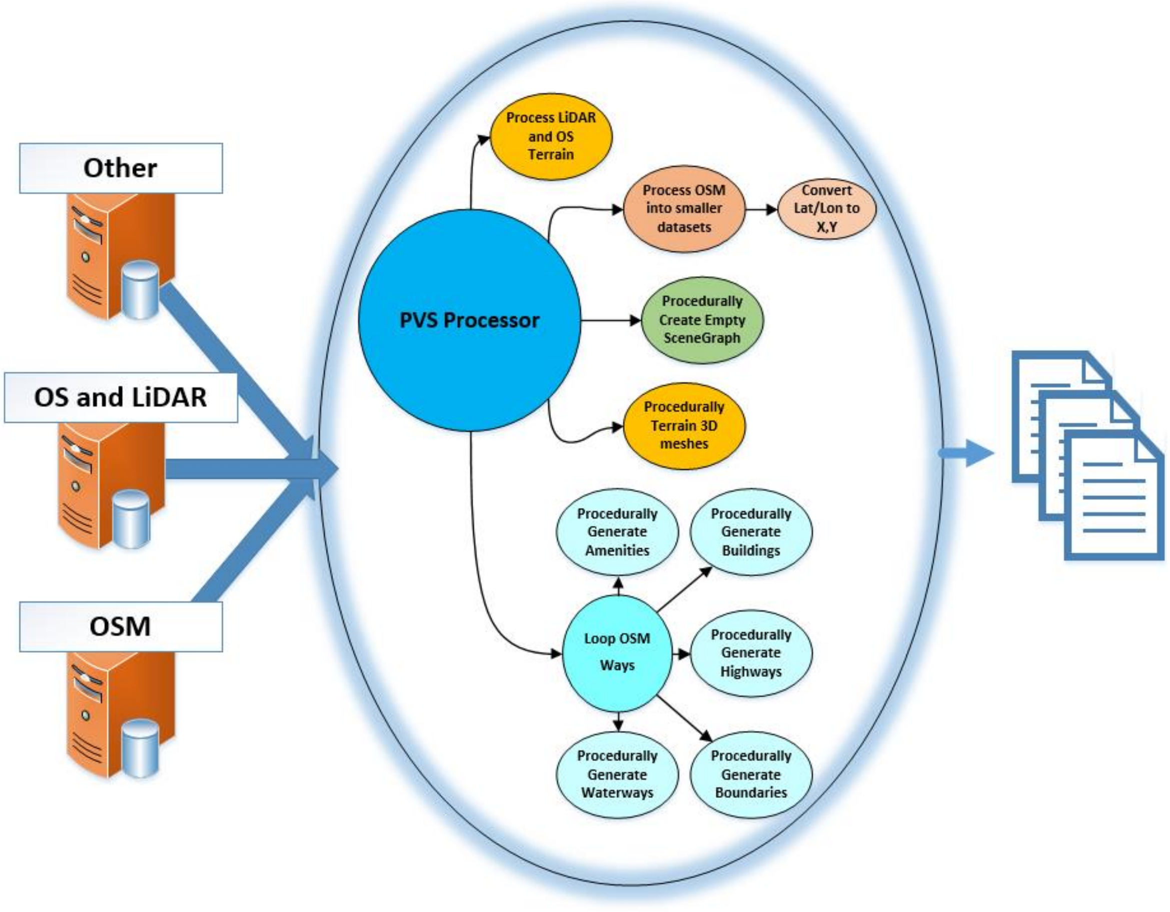

Other
OS and LiDAR
OSM
PVS Processor
Process LiDAR and OS Terrain
Process OSM into smaller datasets
Convert Lat/Lon to X,Y
Procedurally Create Empty SceneGraph
Procedurally Terrain 3D meshes
Procedurally Generate Amenities
Procedurally Generate Buildings
Loop OSM Ways
Procedurally Generate Highways
Procedurally Generate Waterways
Procedurally Generate Boundaries

Figure 37 Input and Output from Processors.

Due to the complexity and novelty of the algorithms we propose, the processes intertwine between each other in a multi-stage procedure as shown in Figure 30 and Figure 32. The reason for a multistage possessor is due to the additional checks needed to combine or infer error prone user/public generated content. To capture the height of a building made from OSM attribute data, which does not contain the height of the building, terrain data from LiDAR data can be used to extract this information. Terrain data can only be used if the terrain data is accurate and has potential errors removed, errors such as missing data points. The terrain datasets need to be processed first, utilising interpolation algorithms to fill small holes within the datasets. If holes cannot be filled, combining LiDAR with OS data can fill the missing data. To improve the terrain datasets produced from combing the OS and LiDAR maps, OSM can further be used to classify areas of the terrain; river ways, coastal lines, highways etc. Using this data, the terrain data can be altered to remove potential errors such as vehicles on the road at capture time, or missing data due to LiDAR not detecting water correctly. The multi-state pipeline is needed for this reason.

Figure 37 shows the inputs of geospatial data and other data sets, processing this data through combining algorithms and PCG techniques to produce the multiple output files needed. The output file for the process is initially an XML file which represents a single PVSNode of the scenegraph. The XML file can then be used within alternative frameworks, game engines, or applications, or be reprocessed to convert them into an optimised binary format for improved loading at runtime.

Figure 38 presents the sequence diagram which shows the interactions between processes shown in Figure 37.

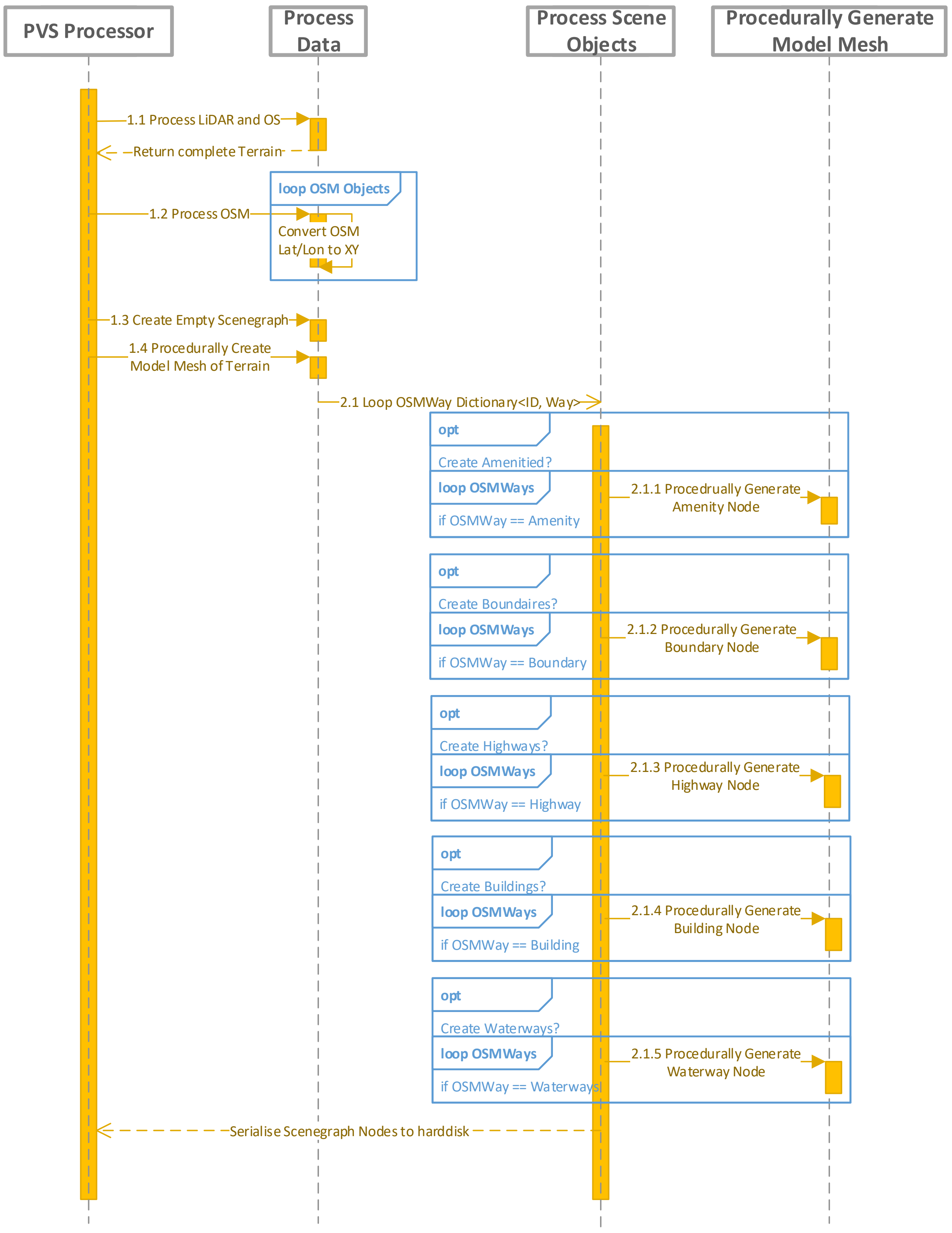

PVS Processor
Process Data
Process Scene Objects
Procedurally Generate Model Mesh
1.1 Process LiDAR and OS
Return complete Terrain
loop OSM Objects
1.2 Process OSM
Convert OSM Lat/Lon to XY
1.3 Create Empty Scenegraph
1.4 Procedurally Create Model Mesh of Terrain
2.1 Loop OSMWay Dictionary<ID, Way>
opt
Create Amenitied?
loop OSMWays
if OSMWay == Amenity
2.1.1 Procedrually Generate Amenity Node
opt
Create Boundaires?
loop OSMWays
if OSMWay == Boundary
2.1.2 Procedrually Generate Boundary Node
opt
Create Highways?
loop OSMWays
if OSMWay == Highway
2.1.3 Procedrually Generate Highway Node
opt
Create Buildings?
loop OSMWays
if OSMWay == Building
2.1.4 Procedrually Generate Building Node
opt
Create Waterways?
loop OSMWays
if OSMWay == Waterways
2.1.5 Procedrually Generate Waterway Node
Serialise Scenegraph Nodes to harddisk

Figure 38 Sequence diagram of PVS data processor

We describe further the algorithms needed and the designs of these algorithms. We also describe the data sets available as the base input for the algorithms, processors, and parsers.

6.3.1 Process LiDAR and OS Terrain

Within this section we will discuss the processes needed to generate a complete terrain data set. We state that complete data set represents a terrain map which contains no missing data values. If a point is uncaptured, its index if populated with the missing data value of -9999. To make the terrain map complete, but may still contain errors, the correction of this value is needed.

6.3.1.1 Combining Maps to fill Missing Data

Within this section, we present the design and algorithm to combine terrain data sets of OS and LiDAR together to create a complete coverage data set. We state that complete means no missing data values are contained within a terrain map. Erroneous data may still be within the map.

The algorithm stated next assumes a base map is available. The base map will be either; a map of lower resolution which has been interpolated to the current resolution, or in the case of the LiDAR DSM, the base map will be the DTM map at the same resolution. The order of base maps needs to be for OS at 2m resolution, LiDAR DTM at 2m resolution, and then DSM at 2m resolution. These terrain files will be the base terrain maps for the proceeding terrain generation techniques.

The algorithm is shown within Figure 39. Within the pseudo algorithm, the variables stated are; user defined iteration count, map width, map height, gap size, and user defined gap size limit. The input data is a 3*3 matrix of the current data maps with the centre map the one being processed.

The user defined iteration count is a value stated by the user before the processing commences. This value iterates how many times the algorithm will process the horizontal and vertical processes. The default value is 10. The map width and map height are defined by the current LiDAR map being processed; 2m, 1m, 50cm, 25cm which corresponds to 500, 1000, 1000, and 2000 for both the width and the height.

The gap size is a running total of the current missing data points. When a point is missing, the gap size variable is incremented and stops incrementing when a none-missing data value is found. If the gap size is less than or equal to the user defined gap size limit, interpolation through the points commences.

```
Start
Input = 3*3 Matrix of LiDAR maps with centre map being the selected map.

if any point is equal to -9999, map is incomplete
    for i < user defined iteration count (default 10)
        int gapsize = 0
        for j < map width
            if j value == -9999
                increment gap size
            if gapsize <= user defined gap size limit && next point != -9999
                for k < gapsize -1
                    Interpolate through gap points k to k+1
                gapsize = 0
        gapsize = 0
        for j < map height
            if j value == -9999
                increment gapsize
            if gapsize <= user defined gap size limit && next point != -9999
                for k < gapsize -1
                    Interpolate through gap points k to k+1
        if map is complete
            return;
if map is NOT complete
    Load in base map and combine to fill missing data points.
    Interpolate map to higher resolution using the 3*3 Matrix of adjoining neighbour maps
    as tangent input point and output points.
End
```

Figure 39 Pseudo algorithm for DTM and DSM LiDAR map interpolation to create terrain data files with no missing data values of -9999.

The algorithm to interpolate and combine terrain datasets is pictured in Figure 40. The black sections in image 1 represent the missing data. Image 2 and 3 represent the iteration of horizontal and vertical checks and interpolation. Image 4 shows that missing data still persists within the terrain map. Image 5 represents the base map which has been interpolated from a lower resolution to the current resolution; in this example, from 2m to 1m. Image 6 represents the combining of terrain data. Image 7 represents the interpolation of the now complete map to the higher resolution. It also brings attention to the need to interpolate between neighbouring maps at the edges.

The importance of the interpolation function to be used needs to be considered. The next section will investigate the appropriate interpolation function to use with this algorithm.

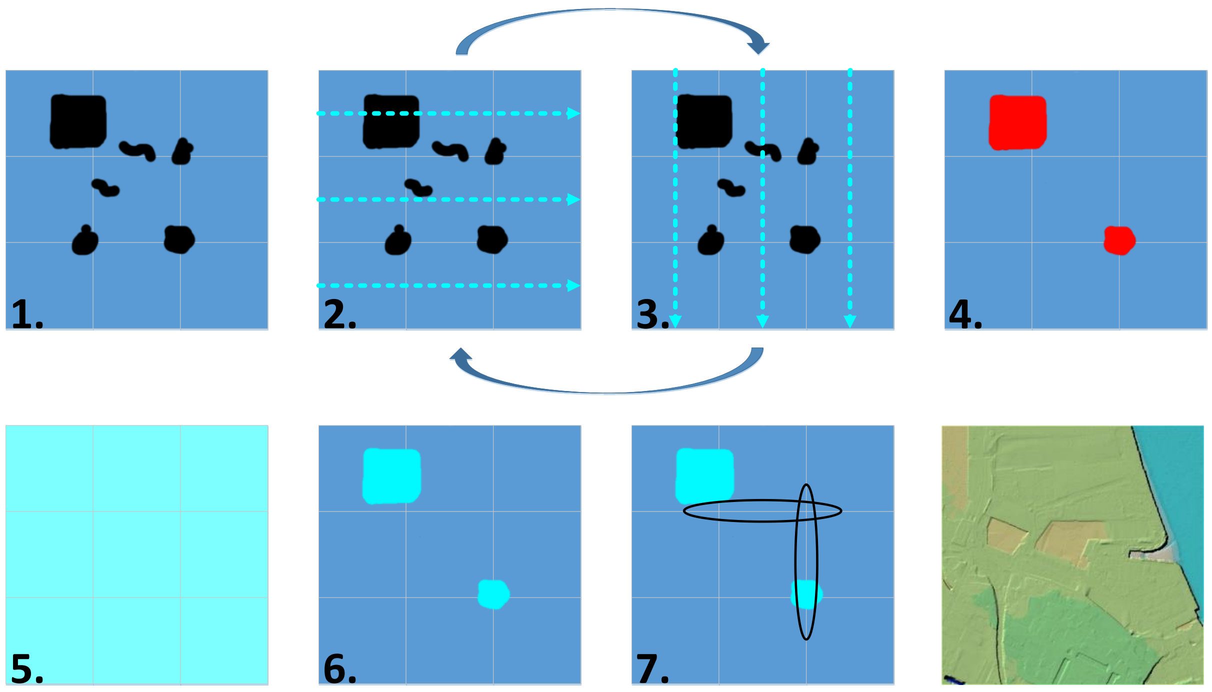


Figure 40 Interpolating and combining terrain maps to create complete datasets.

6.3.1.2 Evaluations of Interpolation Functions to create base terrain data-sets

LiDAR terrain data has issues with being error prone and incomplete. The erroneous data can be removed through hand editing, but creating a complete data set in relation to have not 'missing data values' within a LiDAR map can be generated programmatically. To generate data to replace the missing data values of height values at indices, interpolation between available points needs to take place. Data from external sources may be used to extract data from, but if the data sets are not of the same resolution as the map being checked, interpolation will still have to take place.

Within this section, the comparison between interpolation techniques as to find which is the most appropriate to use for this data set.

The evaluation will consist of utilising a LiDAR map which is complete and satisfies common terrain characteristics; containing water, moderately flat landscape with smooth hills, and some sharp drop-offs. The LiDAR DTM 2m map which contains these characteristics will be used. The total average height will be calculated and used as comparison. An OS map will then be interpolated to the same resolution and point density. The average height will be calculated for the interpolated OS map, and compared against the LiDAR DTM average. Figure 40 shows the characteristics stated are needed and which are found in a majority of the terrain maps within the image on the bottom right.

The experiment compares the interpolation techniques of; quantic, quartic, exponential, Catmull-Rom, quadratic, sinusoidal, cubic, circular, and linear. The equations have 3 functions which are used; EaseIn, EaseOut, and EaseInOut. EaseIn eases the values into the interpolation. EaseOut eases out of the interpolation value. EaseInOut eases in and out through the interpolation between two values. Each function generates different interpolation values between the two values. Figure 41 shows the differences between the functions. Each function may be beneficial for terrain data generation. EaseIn may be useful for creating values which represent the height change between the floor and a wall. EaseOut may suggest the exit between walls to the roof of a building. EaseInOut may be used where the two height points being interpolated represents the side of a tall building.

The benchmarking of the individual functions of each interpolation equation is found. Benchmarking the interpolation functions is created by recording the time taken for each function to be executed 10,000 times and shown Figure 43 and within Table 10.

As well as benchmarking, the comparison between a 1m resolution 1000^2-point map with has complete coverage, i.e. it has no missing data values, and a map which has been interpolated from a 2m resolution 500^2-point map to a 1m resolution 1000^2-point map has been recorded for each interpolation equation. The average height error in meters for the complete map is calculated. This is found by summing all values, and dividing the summed value by the total number of points.

The comparison between interpolation functions are found within Table 11 and Figure 43.

From the results, we can infer that the Quintic equation is the most accurate, but also uses the most calculations. The use of this function should be kept to processes where accuracy is more important than speed, and therefore a less intensive equation may be appropriate.

We have included Catmull-Rom interpolation into the experiments. The functions available do not include the EaseIn, EaseOut, or EaseInOut. It is included due to the importance of the entering and leaving tangents angles of the interpolation as shown in Figure 44.

For terrain generation, Quintic, or Catmull-Rom interpolation is sufficient.

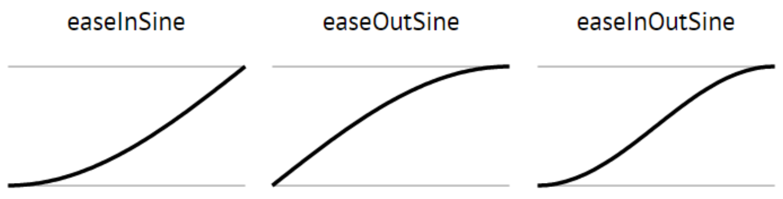


Figure 41 EaseIn, EaseOut, and EaseInOut interpolation functions.

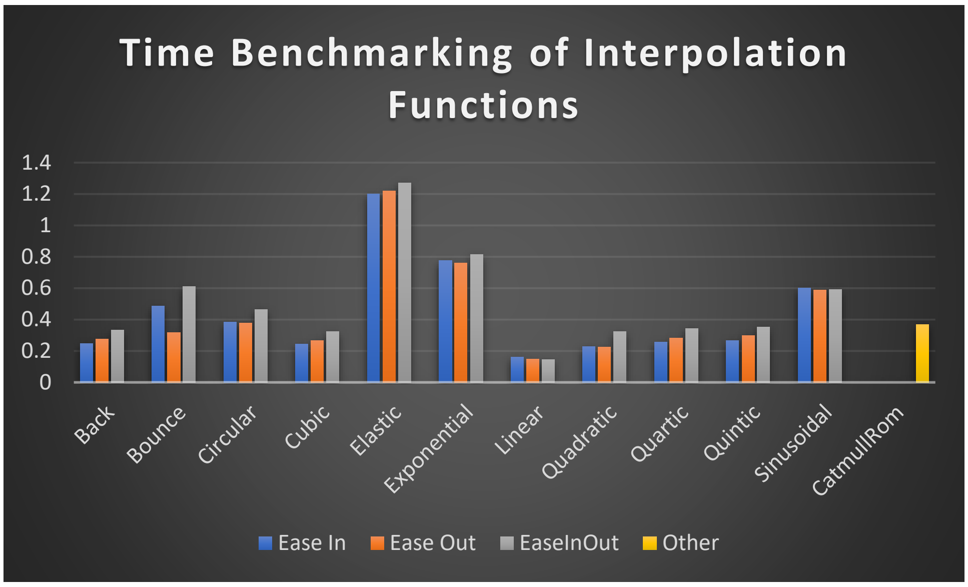


Figure 42 Benchmarking of interpolation functions from the XNATweener library bar chart.

Function	Ease In milliseconds	Ease Out milliseconds	EaseInOut Milliseconds	Other Milliseconds
Back	0.2491291	0.2754739	0.3353423	0
Bounce	0.4878453	0.3152747	0.6101821	0
Circular	0.3842962	0.3789251	0.4644622	0
Cubic	0.2414599	0.2671496	0.322852	0
Elastic	1.2005921	1.2189915	1.2708431	0
Exponential	0.7750527	0.7597306	0.8145901	0
Linear	0.1602594	0.1478687	0.1437446	0
Quadratic	0.2287326	0.2267471	0.3215284	0
Quartic	0.2572451	0.2809173	0.3421836	0
Quintic	0.2671735	0.2986885	0.3517596	0
Sinusoidal	0.6025026	0.5901671	0.5922018	0
Catmull-Rom	0	0	0	0.368244

Table 10 Benchmarking of interpolation functions from the XNATweener library.

Interpolator	Error in metres
Quintic	0.06115
Quartic	0.06162
Exponential	0.06176
Catmull-Rom	0.06191
Quadratic	0.06223
Sinusoidal	0.06224
Cubic	0.06224
Circular	0.06237
Linear	0.06294

Table 11 Interpolation results of 50m resolution OS map to 2m resolution LiDAR map.

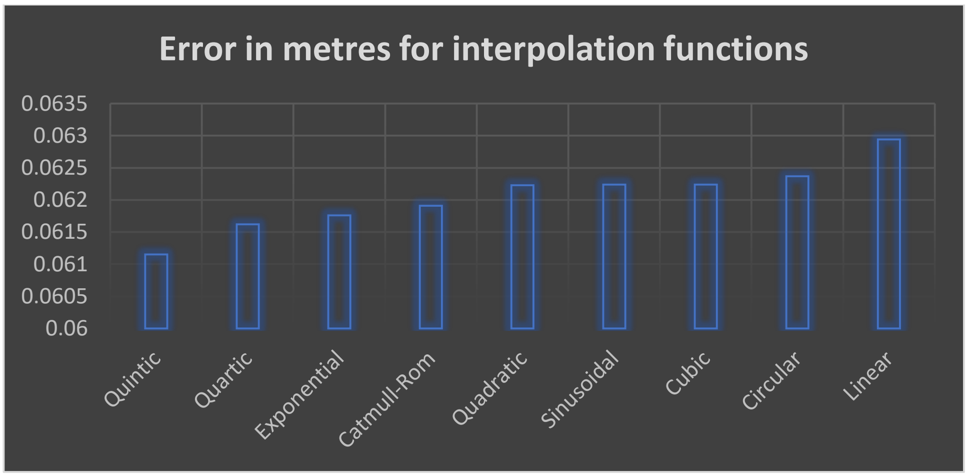


Figure 43 Graph showing the total mean error of interpolation functions used to interpolation a 50m resolution OS map to a 2m resolution LiDAR map.

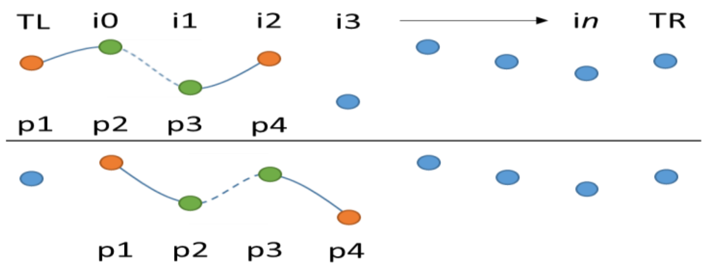


Figure 44 Catmull Rom interpolation representation.

6.3.2 Processing OSM

While processing OSM data, multiple procedures must be undertaken.

Each OSMWay will have its OSMNodes analysed and classified. The OSMTag attributes will be extracted from the OSMWay, and the OSMNodes referenced by the OSMWay. Using the attributes, PG techniques are utilised in tandem with the attributes to classify, and create content.

The algorithm for the OSM processor is shown in Figure 45. The input for this algorithm is the user generated content schema which states which assets need to be processed; buildings, highways, amenities etc. as well as the OSM XML file which is parsed into runtime objects. The output of the algorithm is an array of serialised XML files. Each file represents a single node of the scenegraph containing the model mesh data, and the spatial and visual parameters.

```
Start
Procedurally generate OS name references
Import user generated content schema file
Initialise framework attributes
Import OSM XML file
for-each OSM Node
    Convert Latitude and Longitude to X, Y, Z coordinates
    Extract OSM Nodes and populate dictionary<ID, OSM Node>
    Check for duplicated objects
    Extract OSM Ways and populate dictionary<ID, OSM Way>
    Check for duplicated objects
Calculate bounds of OSM map
Create Scenegraph from OSM map bounds
Create OSM categorisation sub-trees
Generate terrain model meshes
for each OSM Way in Dictionary<ID, OSM Way>
    if OSM Way type == 'Boundary'
        Generate boundary asset
    if OSM Way type == 'Building'
        if asset is on hard drive
            Parse content
        else
            Generate building asset
    if OSM Way type == 'Highway'
        if closed, generated round-about
        if open, generate highway
    if OSM Way type == 'Amenity'
        Generated Amenity asset
    if OSM Way type == 'Waterway'
        Generate Waterway asset
Serialise Scenegraph to disk.
End – output are nodes of the scenegraph containing all information pertinent to that
node.
```

Figure 45 Pseudo algorithm for the processing of OSM Data

The generation of OS name references is needed for the lookup of neighbouring maps. The names are used in a later process to calculate the bounding box of OS tiles used for categorisation within the scenegraph. The pre-generation of bounding boxes for full UK coverage is unwise due to the large amount of memory needed, and processing time needed to procedurally create all bounding boxes at the multiple levels of OS mapping schema. The names will be used as the input to procedurally generated bounding boxes. This improves memory and processing consumption.

The user generated content schema file is a text file which contains the ID of an OSMWay which the user wishes to swap the procedurally generation techniques for user generated content; i.e. the procedurally generated building model mesh is substituted for a pre-made 3D model mesh. The issue with this is the user generated 3D model mesh content may not be set to the correct translation, rotation, or scale. This data is included along with the user generated content name on disk. This allows for user generated content to be compiled and parsed into the scene correctly for visualisation at runtime. This removes the need for PG of assets. Depending on the complexity of the user generated model mesh, processing time may be increased, but the visual realism will be improved. The OSMNodes and OSMWays are added to individual Dictionary<*Key*, *Value*> data structures. This is used for quick lookup due to a Dictionary having a BigO time of O(1) complexity. The checking of duplicated data is also applied during the insertion of elements to the dictionary data structures. As stated, OSM has procedures to check for duplication of data, or objects within the database which reference the same unique ID. Our check will add OSM objects to the dictionaries on a first-come first-served basis. The conversion from Latitude and Longitude to X, Y, Z coordinates are calculated and added to the OSM objects. This reduces potential extra calculations in later processes.

Due to OSM maps having OSM objects outside of the selected map area, this creates a dilemma; only create a scenegraph hierarchy which contains objects within the selected bounds, or increase the bounds to include all objects. We choose the latter as to not miss information or objects which may be useful. An example of this may be a boundary which can be used to classify unclassified assets.

Using the bounds, the Scenegraph bounding hierarchy can be created. Attached to the leaf nodes of the bounds, the OSM categorisation subtrees. Using the bounds of the scenegraph, the terrain model mesh objects can be created from the terrain input data. Once PG of the terrain model mesh objects is completed, they are added to the subtree. Figure 46 shows the pipeline for creating a 3D model mesh for terrain.

This technique splits the Scenegraph into two sections; the first is the spatial bounds, and the latter is the OSM and terrain categorisation. This is explained further in a later section. In essence, this design choice can improve search techniques by spatial queries, and categorisation queries.

The looping of the OSMWay objects is processed next. Within the loop, each OSMWay is checked. The checking of the OSMWay type is done in a specific order. The boundary objects are processed first. This is to allow the classification of assets in a later process by querying the location of an asset, and whether it intersects with a boundary. If true, then the unclassified asset will be classified further. For example, a building which has no type, may lie inside a 'residential' boundary asset, thusly the building asset can be inferred to be of type 'residential'.

The building assets are created next, checking if a user generated model mesh object is on disk.

We state that only leaf nodes of the scenegraph should contain model mesh objects. We also state that nodes of the tree which happen to be leaf nodes and do not contain a model mesh object should not be removed due to its importance as a lookup node.

Due to the depth of the PG of 3D model assets from OSM and other input data, we extend the discussion in the following section about the procedure with PG techniques to be used.

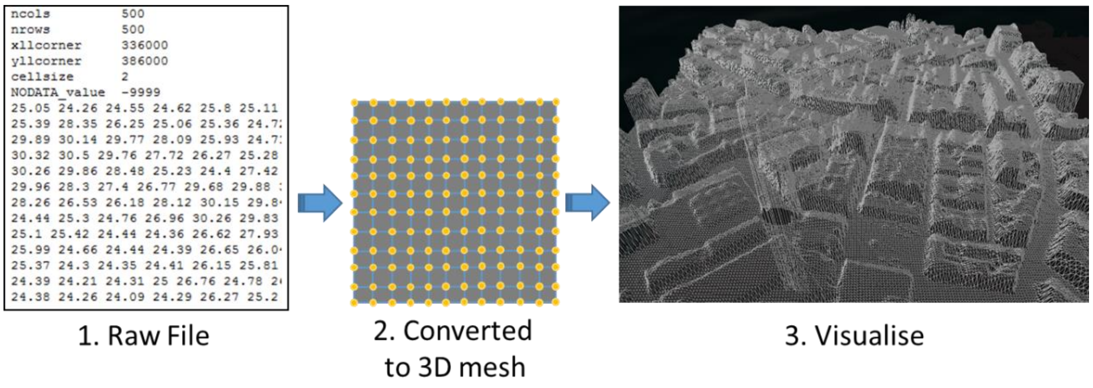

```
ncols         500
nrows         500
xllcorner     336000
yllcorner     386000
cellsize      2
NODATA_value  -9999
25.05 24.26 24.55 24.62 25.8 25.11
25.39 28.35 26.25 25.06 25.36 24.7
29.89 30.14 29.77 28.09 25.93 24.7
30.32 30.5 29.76 27.72 26.27 25.28
30.26 29.86 28.48 25.23 24.4 27.42
29.96 28.3 27.4 26.77 29.68 29.88
28.26 26.53 26.18 28.12 30.15 29.8
24.44 25.3 24.76 26.96 30.26 29.83
25.1 25.42 24.44 24.36 26.62 27.93
25.99 24.66 24.44 24.39 26.65 26.0
25.37 24.3 24.35 24.41 26.15 25.81
24.39 24.21 24.31 25 26.76 24.78 2
24.38 24.26 24.09 24.29 26.27 25.2
```

Figure 46 3D terrain model mesh creation algorithm.

6.3.3 Procedurally Generate 3D Assets

Within this section we will discuss the algorithms needed to generate the individual assets from OSM data.

We will start by presenting the design for the algorithm which creates the PVSTag objects which hold the attributes extracted or inferred from OSM. Next we discuss the generation of the boundary model mesh objects. Boundaries are created first to allow classification of other assets if those other assets lie inside or near a boundary. After this, we state the processes to generate the buildings, highways, amenity locations, and water ways.

6.3.3.1 PVSTag

3D model assets will have a PVSTag object. The PVSTag object will contain the attributes for the asset. The attributes will be pertinent to the specific assets; a 'building tag' for a building, a 'highway tag' for a highway etc. The attributes will have default values which will be overwritten within the generation process. The default values will need to be consulted by a domain expert. For example, the default value of a single floor height of a building is 3.1meters according to the *Council on Tall Buildings and Urban Habitat*[42]. Each PVSTag will have similar default data; number of lanes for a highway, the textures to use, shader parameter settings etc.

Figure 47 shows the UML diagrams with accompanying Reader objects for the OSMTags. A Reader object is used to read in and parse the binary data at runtime. An example of the attributes for a building are; Address City, Address Postcode, Name, Alternative name, Capacity, Cladding, Fireproof, and many more.

The PVSTagInterface is used as an interface object so a PVSNode can store any type of PVSTag.

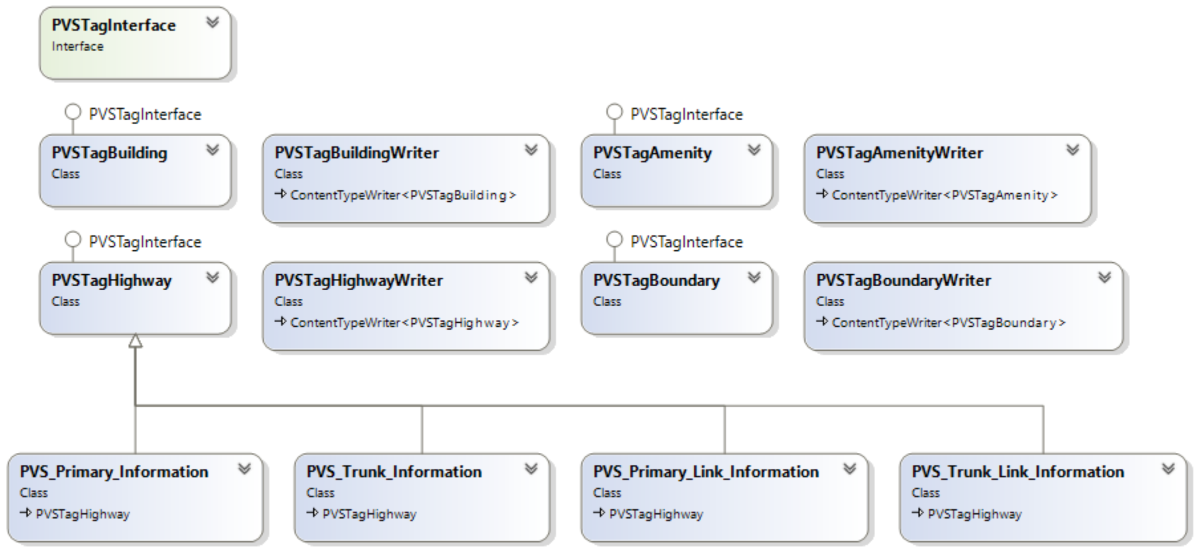


Figure 47 UML Diagram of OSMTag objects attached to nodes and models.

6.3.3.2 Triangulations for model meshes

The use of triangulation software is needed. Many mesh objects require the use of triangulation to generate model mesh objects created from an arbitrary array of points. Figure 48 shows the process of the triangulation needed. The input of an arbitrary of points extracted from OSM, the triangulation algorithm will output the indices buffer, and the vertex buffer. The indices are used as optimisations within the rendering pipeline.

[42] http://www.ctbuh.org/TallBuildings/HeightStatistics/HeightCalculator/tabid/1007/language/en-US/Default.aspx

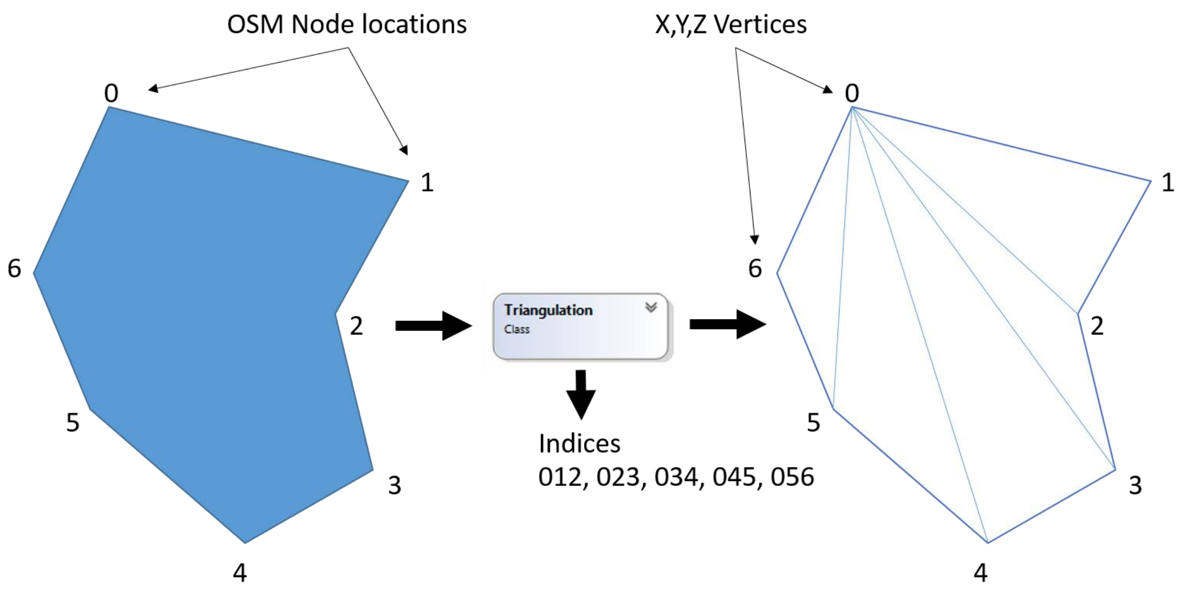


Figure 48 Triangulation input of arbitrary point and output of indices array, and vertex array.

6.3.3.3 Boundaries

The boundaries are to be generated by looping through the OSMWay nodes and creating a polygon triangulated by the triangulation library. During the generation process, the boundaries will have the attribute extracted from the OSM dataset.

6.3.3.4 Buildings

The building assets will have many checks within the algorithm to categorise the type of the building. There are two main buildings we check for; a single building which is generated with a single OSMWay, and a building which is made with multiple OSMWays by checking the OSMRelation. Buildings which are made of multiple OSMWays have multiple parts to the building.

The process is as follows;

- Between two points of the OSMWay, extrude a vertical polygon to represent the side of the building. Repeat this process through each point within the OSMWay.
- While creating the polygons, texture UV coordinate will also be generated by measuring the length between the two points. This will wrap a texture around the building.
- A triangulation algorithm will be utilised to generate the roof structure.

Utilising these processes, basic building model meshes can be created.

The type of a building is stored within the OSMTag. Buildings are made of multiple OSMNode objects or multiple OSMWay objects. In this situation, a building may have multiple classifications, or various data which can be inferred. For example, a building which is classified as religious can have multiple religious' descriptors; Christian, Catholic, Hindu etc. A counting system is needed to count the descriptor tags, and apply the highest value as the dominant type for the building. This should have effect on the diffuse texture used on the model mesh object. In short, it counts how many descriptor tags are of type Christian, Catholic, Hindu and the one with the highest count sets the buildings type, and changes the diffuse texture embedded into the model mesh.

6.3.3.5 Highways

The highways of OSM are rigid and angular due to the single line nature of the ways. To produce highway model mesh objects, the sides of a highway will have to be generated. The OSMWay has the node locations within the centre of the highway structure it is representing. Using this, the width of the highway can be queried within the OSM data, or inferred by calculating the width by the number of lanes.

Figure 49 show the process needed. Step 1 is the OSMWay representation of the angular highway. Step 2 is the interpolation of the points from the OSMWay. We have chosen Catmull-Rom interpolation given its entering and exiting tangents. Quintic interpolation a 0.001% improvement on Catmull-Rom, whereas Catmull-Rom improves on processing speed. Step 3 shows the points. Step 4 is the production of the sides of the highway structure. The process in step 4 is repeated for the length of the interpolated line.

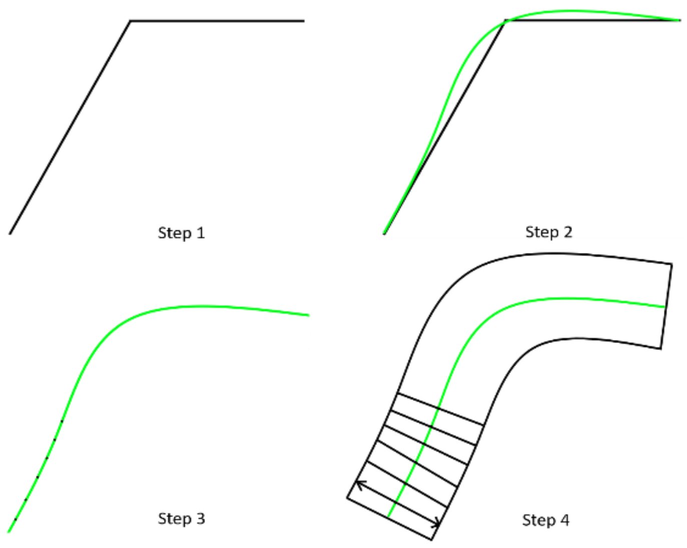


Figure 49 Highway mesh generation. Step 1 is OSMWay representation. Step 2 is the result of the interpolated points. Step 3 shows the points generated from the interpolation. Step 4 generates the perpendicular sides of the highway.

6.3.3.6 Amenities

The amenities are locations within the world of particular significance. The locations will be represented by procedurally generated models at runtime. This is due to the number of amenities found within built up city environments, and the fact that the model meshes to be used are primitive and thusly create extra data. Rendering techniques such as instanced-rendering can be used to render the same primitive model at multiple locations, reducing the memory consumption needed for duplicated data.

6.3.3.7 Texturing Polygons

To allow texturing of polygons, procedural methods are needed to calculate the UV coordinates the polygon should have to overlay textures clearly and accurately. The process is pictured within Figure 50. Step 1 is to the vectors of the polygon which are furthest left, furthest right, furthest top, furthest bottom. Using these locations, the UV coordinates can be generated by negating the positions from one another. Using the UV coordinates at rendering time, the texture can be overlaid as shown.

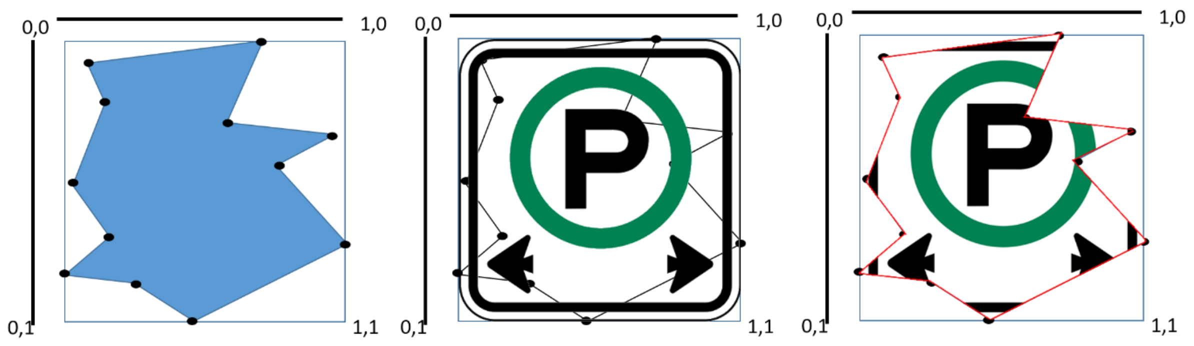


Figure 50 UV coordinate generator process.

6.4 PVS Model Object

We have stated in the specification that a model mesh must be capable of storing multiple mesh parts. This is to allow flexibility and manipulation of appropriate parts of a model, and the parameters of the materials for each model part. The parts will store the vertex buffers and index buffers used for rendering. Each model part stores a material object which stores the rendering parameters; ambient colour, diffuse colour, specular colour etc.

The attributes which the model will use are: bounding box, bounding sphere, an array of model parts, a name attribute, and a reference to the node it is attached to. Additional functions are needed to disperse data to each of the model parts, and create connections between the model parts and the node the model is attached to.

The UML Class diagram within Figure 51 shows the structure of the model class needed. We have emitted the functions and will list them here with a brief description of what the function does.

The data and data-structures which need to be passed are; effect object, material object, referenced or unreferenced, graphics rendering state, referenced or unreferenced, light descriptions, referenced or unreferenced.

Other functions are also needed to reset objects to the original values; materials, light descriptions, shallow copy, and deep copy. Other functions may become apparent during the implementation phase.

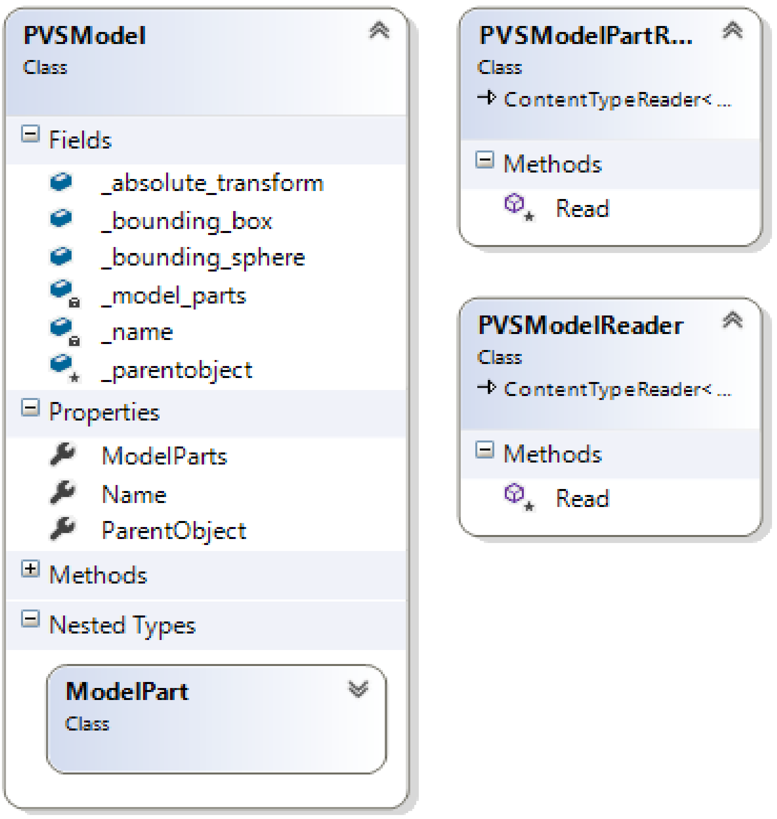


Figure 51 UML Class diagram of the PVSModel and nested model part object.

6.4.1 Model Part

A model part represents different parts to a model; sides, rooftop, window etc.

The justification of having model parts is for visualisation purposes. An asset such as building within real-world scenes are made of various parts and structures, and thusly they should be represented the same. This also allows multiple visualisations and animations to be applied to a single model but on separate model parts.

Figure 52 shows the UML Class diagram of the properties and functions within the class. The model part may contain duplications of attributes and references such as the original material object, and the current object. This allows for temporarily modifying the material attributes but revert to the original. This is imported as to limit loading of original data, but will increase memory usage.

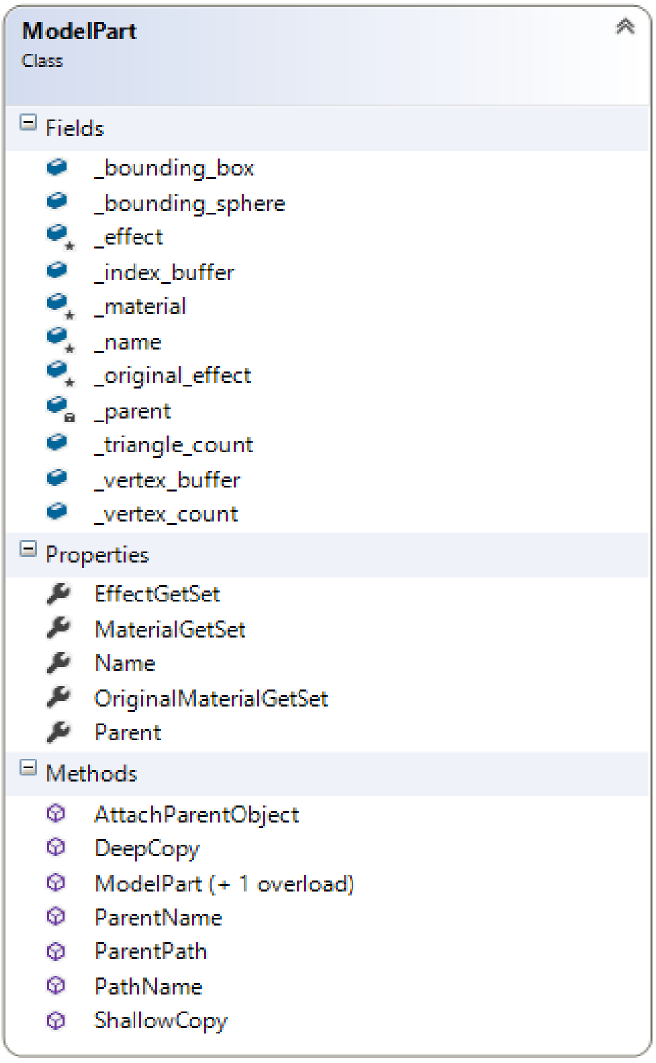


Figure 52 UML Class Diagram of the Model Part object.

The property definitions are show in Table 12.

Fields	Definition
Bounding Box	The cuboid bounds generated from the vertex points.
Bounding Sphere	The spherical bounds generated from the vertex points.
Effect	The effect used to render the object on the GPU. This is built of vertex shader, and pixel shader functions.
Original Material	The immutable material object. In some cases, the material parameter may be required to change and be saved to improve future renderings.
Material	The mutable material object.
Index Buffer	The array of indices used for optimised rendering.
Vertex Buffer	The array of vertex object used for rendering the model mesh.
Parent Reference	The reference to the parent model. It allows passing of messages from material to parent or the rest of the scenegraph.
Vertex count	The count of how many vertices are in the model. Used for monitoring.
Triangle count	The count of the triangles within a model. Used for monitoring.

Table 12 Model part properties and definitions.

6.4.2 Model Material

The material which is used to pass parameters to the GPU effect file will store the parameters needed for rendering. To create efficient updating of the GPU, helper functions will be designed. Various structures which combine parameters are created to ease development and categorisations of similar parameters. The material also contains a list of controllers. These controllers can be attached to the material to modify the parameters. Default controllers are needed which create a connection between the material and the camera of the scene to check when the view and projection matrices of the camera has been changed. The controller will force an update of the relevant parameters, and pass the parameters to the GPU. See section 6.4.2.3.

6.4.2.1 Material Light Description

Parameters passed to the GPU represent a light; position, direction, colour, spot angle, constant attenuation, linear attenuation, and quadratic attenuation. A structure is needed to contain these parameters for querying and passing to material objects of models.

A Light Description structure is used to contain these parameters, as well as others; light type, and whether it is enabled. The UML class diagram is pictured in Figure 53.

The light type is an enumeration of two parameters; directional or spot. A point light is a commonly used light structure. The equation for the point light can be collapsed into a spot light, thusly, a dedicated point light is unnecessary.

To accompany the light description structure is a static helper class. This class will contain pre-made light descriptions. The premade light description will consist of; red light, blue light, green light as a proof of concept. Future users of the framework can create additional light descriptions.

Passing the light description parameters to the GPU for rendering, the individual parameters are extracted and passed.

We have stated the framework is to only utilise 3 lights, and the light type depicts the structure of the IASF. An algorithm is used to order the three lights into an ordered list from the possible combinations.

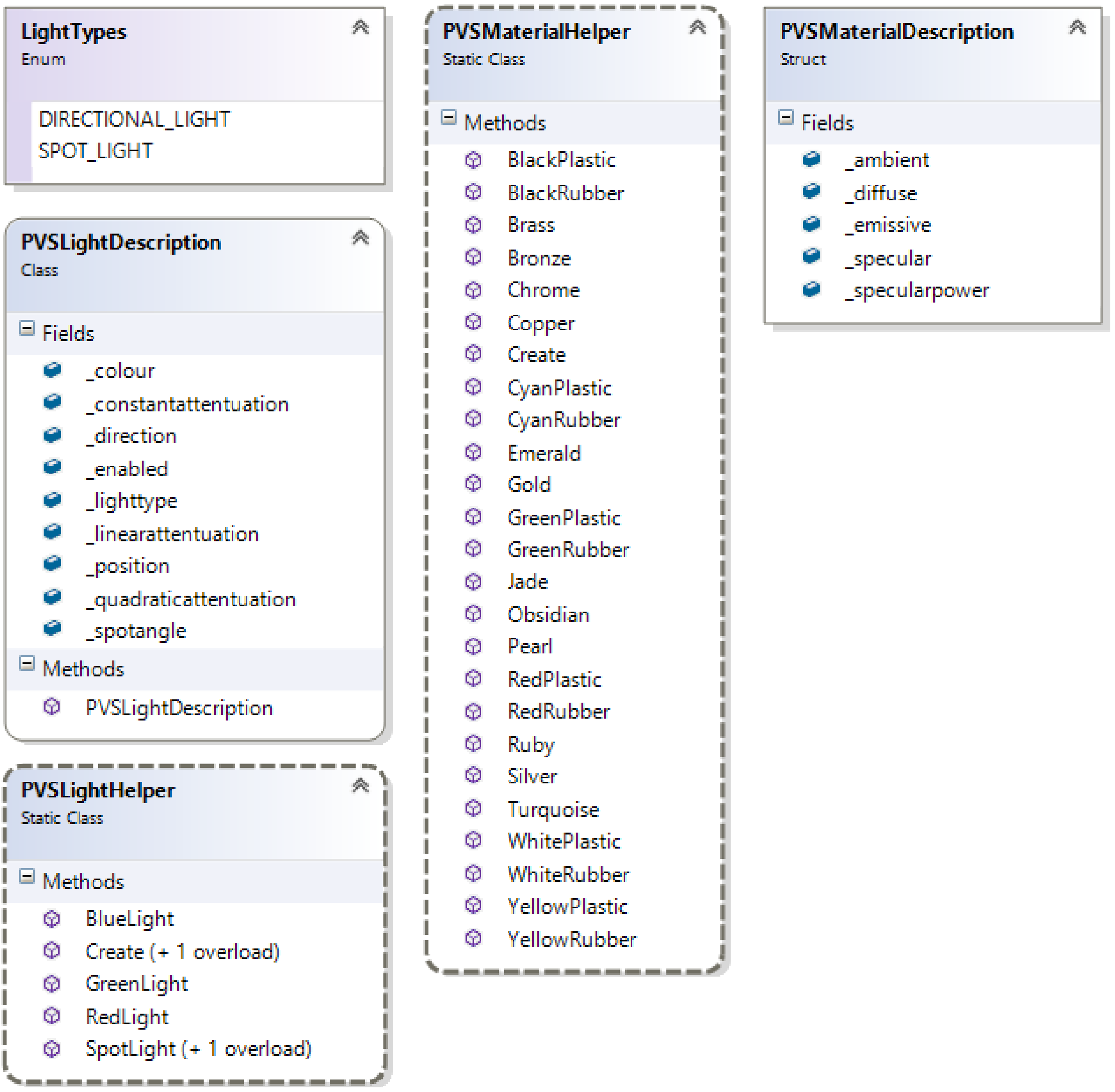


Figure 53 PVSLightDescription and light description helper class used to contain the parameters of a light used for rendering. And the material description and helper class.

6.4.2.2 Material Description

As with the light description explained in section 6.4.2.1, a material description is to be utilised to ease the development and the dispersal of material parameters. The UML class diagram for the material description and helper class is shown in Figure 53.

The static helper class is used for the creation of common colours and materials e.g. black rubber, chrome, red rubber etc. These materials not only state the visual parameters of an object but also the material matter of an object. This is a basic version of PBR.

The use of the material description is to change one or more objects material properties with a single update. For example, a material which modifies the colour of model mesh object to red can be used on multiple objects within a scene. In the relation to buildings, none-fireproof buildings can be modified with a red material description.

6.4.2.3 Material Controllers

To allow the modification of a material, and also remove the need to update materials which do not need modifying, thusly saving small amounts of processing each frame, the use of controllers which target the needs of materials are needed. Each material object will contain an array capable of storing zero or more controllers. The controllers will allow the procedural animation, interpolation, and modification of all parameters of the material. The controllers will have a reference to the material object it is attached to. To reduce reference calls between additional objects from the scenegraph, the model object, model parts, and the material object, a reference to the node which the model and model parts are attached to may be needed. This will allow the skipping calls between objects.

The controllers are used to update parameters of the material. The class utilises an interface class which states that a controller will need an attached object, i.e. the reference to the material it is connected too. The use of the interface is the ability to store multiple types of controllers within a single array structure. It will also need to create an update method. The class is then inherited by specific controller objects. The connection between the material controller and other objects is the notification that the camera has been moved. This will trigger the updating of the parameters such as the eye position, view direction, and the WorldViewProjection (WVP) matrix used to render the model mesh object in the virtual world. An example of the connections is the 'ShaderWorldDirtyHookUp' controller. This controller has a connection to the material it is attached to, the material has a reference to the parent node object it is attached to (PVSSpatial). Within the PVSSpatial class, it has a Boolean flag which states whether the nodes translation, rotation, or scale has been modified. If it has, a 'Dirty' flag is set to true. Within the update of the controller, this flag is checked, and if true, then parameters within the material will be recalculated, e.g. its lighting parameters, and view and projection matrices. The same technique is used for the ShaderCameraHookUp controller, but that looks for a static flag within the camera library to check if the camera has moved. If true then the view and projection matrices are recalculated. Other parameters are also updated such as the view direction, and the cameras position.

This technique allows for animations and interpolations created using the Interpolation/Tweening library, to be attached to any appropriate datatype of the material object; allowing for any property to be animated.

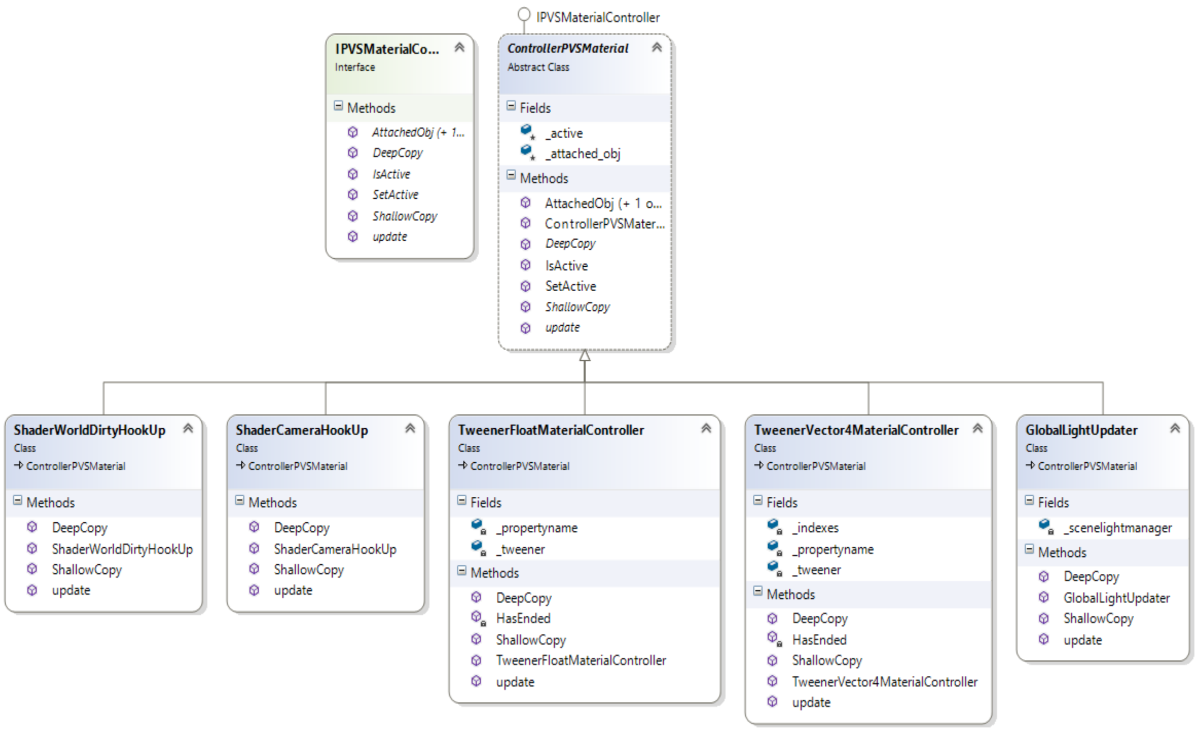


Figure 54 UML Diagram of Material Controllers

6.5 Scenegraph Design

We discuss the design for the scene graph. The scene graph will be utilised within system 1 and system 2 of the PVS pipeline.

After deliberation into the best structure to store objects within the virtual scenes we have concluded that a tree structure is needed. The tree will use the OS reference scheme to subdivide the UK into manageable 1km^2 tiles. Within each of these tiles, the objects created from the OSM data-set will be placed within this branch in a specific order. The objects created from OSM will be organised by specific properties. The scene graph can be thought of as two separate scene graph structures which are combined together. The first part of the scene graph is the bounding volume hierarchy (BVH), while the top half is a collection of objects pertinent to each tile.

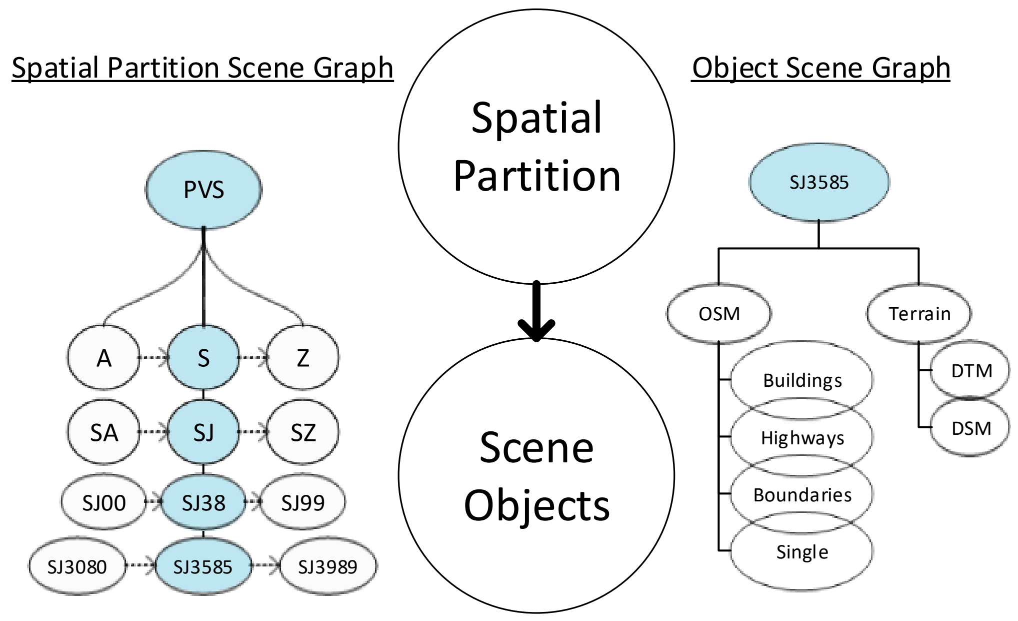


Figure 55 Spatial and Object Scene Graph structures

This divides the tree structure into manageable branches categorised by common features. A branch of the scene graph will be used for buildings, and that branch will have branches containing buildings specific to emergency services; hospitals, police stations, ambulance stations. The reason for this is for increased branch traversal speeds. Knowing the location of specific nodes within the scene graph structure will allow increased speeds of manipulation of emergency service centres. Additional nodes can be added if needed. The other categories which we state are needed are;

- Buildings; contain branches for emergency buildings and a branch for all other types of buildings – normally unclassified buildings. This can be seen in Figure 58.
- Highways; contain branches for highways of type: roads, link, paths, special, railways, and unclassified. This branch is shown in Figure 59 and the types of highways are shown in Figure 60, Figure 61, Figure 62, and Figure 63
- Boundaries; contain emergency boundaries such as hospitals and police areas, waterways such as rivers, streams etc., and *other* types of boundary such as parking.
- Single; this branch will contain a collection of single node locations which provide information about object within a world. This can be a landmark, or a buildings details.

The terrain will contain its own dedicated branch. This branch will contain the two types of LiDAR maps; DTM and DSM. These will also contain the different resolutions of maps from; 2m, 1m, 50cm, and 25cm.

The branching between the OSM assets and the terrain assets held within a 1km^2 tile are shown in Figure 57.

The base of the tree structure will represent the OS reference scheme as stated. The base node will be named ‘PVS’ and this will act as the *World* node which every other node is branched from. The child branches in relation to the OS scheme represent the same projections; each child of the *World* node represents a 500km^2 area, the same as the OS reference scheme at its lowest level of projection. Covering the UK is a 5*5 grid of 500km^2 areas. Each of these grid locations are subdivided into a 5*5 grid of 100km^2 areas. Each of these is then subdivided into 10*10 grid of 10km^2. And finally, each of these is subdivided again into 10*10 grid of 1km^2 areas. The structure for the OS representation in tree form is visualised in Figure 56. For simplicity sake, we do not display all nodes possible for the tree because this will be 6,250,000 nodes at layer 4. A total of 6,286,151 are needed to create full coverage. We show a subcategory. The letter ‘I’ is not used with the OS reference scheme, thusly our representation assumes this fact.

With this scenegraph design, a world of assets generated from OSM, and terrain from OS and LiDAR can be categorised, and used within our framework. The scenegraph structure is custom for this type of data but can be used within other frameworks and game engines within minimal of changes, if any changes.

To generate the scenegraph, a factory design construct will be utilised. Within this construct, functions are used to create and combine empty nodes of the scenegraph. When we state empty we mean nodes which do not contain a model mesh object.

The factory construct will utilise procedural functions. Creating a scenegraph which contains all nodes that contain the whole UK is unneeded. Passing the bounds of the selected OSM region to the procedural function for the factory to only generate the nodes necessary.

Before processing the OSM map, lookup tables of 2D array structures are generated. The multiple lookups represent the different resolutions of OS; 500km^2, 100km^2, 10km^2, and 1km^2. The look ups are only the names of the tiles. The bounding boxes are to be procedurally generated to save storage and processing.

Processing the selected OSM map, the bounds of the map are checked. The check loops through the nodes within the map file, and will check for the furthest left, furthest right, furthest top, furthest bottom by converting the longitude and latitude to OS X, Y coordinates. A bounding box is created. It is this bounding box which is checked against the procedurally checked OS reference maps. The algorithm for creating the list of tiles which the OSM bounds collide with is listed next;

- Loop through the array of 500km^2 entries. Procedurally generate a bounding box from the name of the tile.

- Check if the OSM boundary intersects the procedurally generated tile bounds.
- If this is true, then do the same procedure for the 100km tiles of the 500km^2 tile it has just intersected with. Repeat this through the 10km^2, and 1km^2 tiles.

The OSM scenegraph branches are to be generated within single functions. A function to generate the buildings branch, a function to generate the OSM highway branch etc.

Using multiple function calls we can string the function calls together to generate a scenegraph. If no buildings are contained with the OSM selected area, a branch does not need to be generated, and thusly we don't need to store or process empty nodes.

The total number of nodes within each 1km^2 tile which depicts the terrain and OSM assets, is 93. This adds the total number of nodes to (6,250,000 * 93) + -36,151 = 581,250,000. This is a minimal number of nodes needed for the UK.

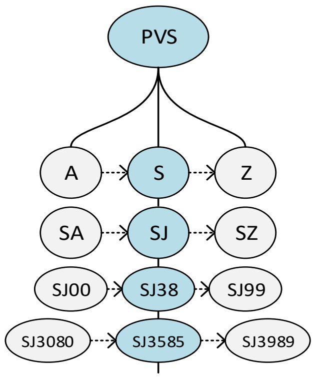


Figure 56 Scene Graph for OS representation and scene subdivision

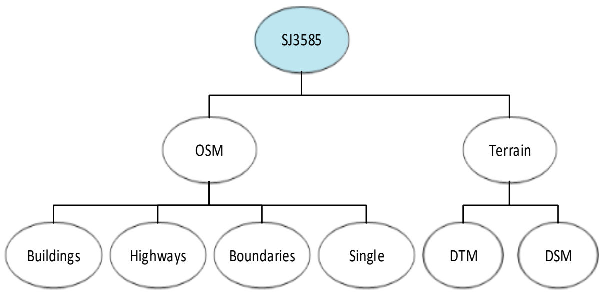


Figure 57 Scene Graph representation for a 1km node. Contains root of the OSM scene graph and the Terrain scene graph.

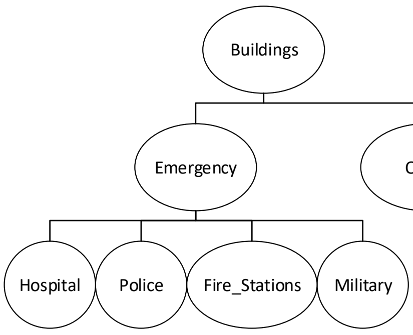


Figure 58 Representation of the node hierarchy for the Buildings branch.

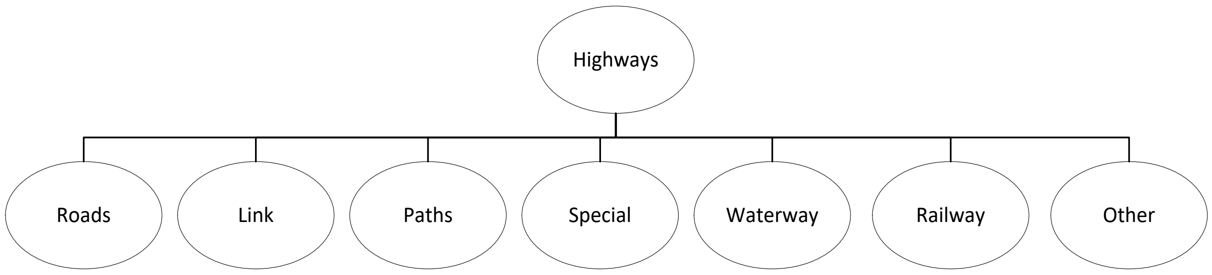


Figure 59 Representation of the node hierarchy for the Highways branch.

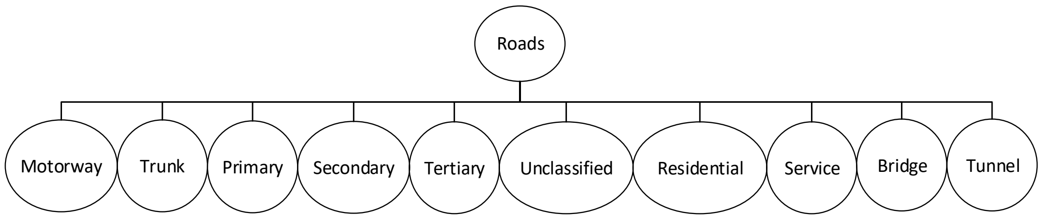


Figure 60 Representation of the node hierarchy for the Roads branch.

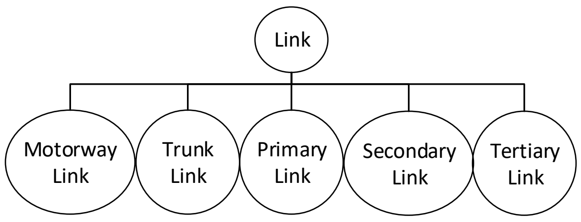


Figure 61 Representation of the node hierarchy for the Link branch.

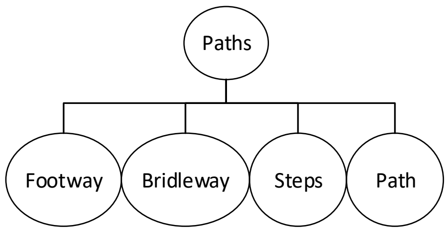


Figure 62 Representation of the node hierarchy for the Paths branch.

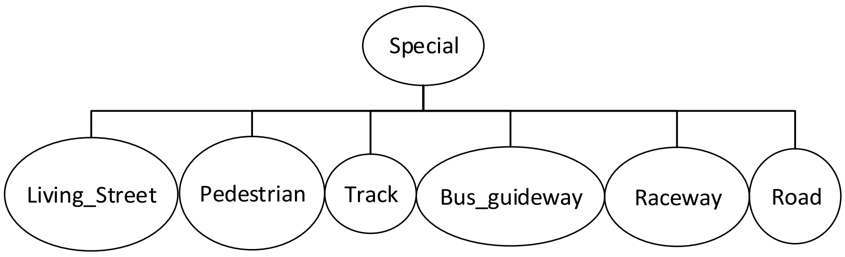


Figure 63 Representation of the node hierarchy for the Special branch.

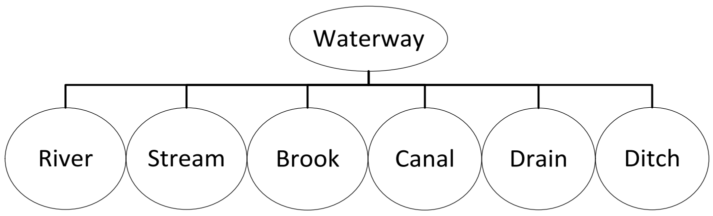


Figure 64 Representation of the node hierarchy for the Waterway branch.

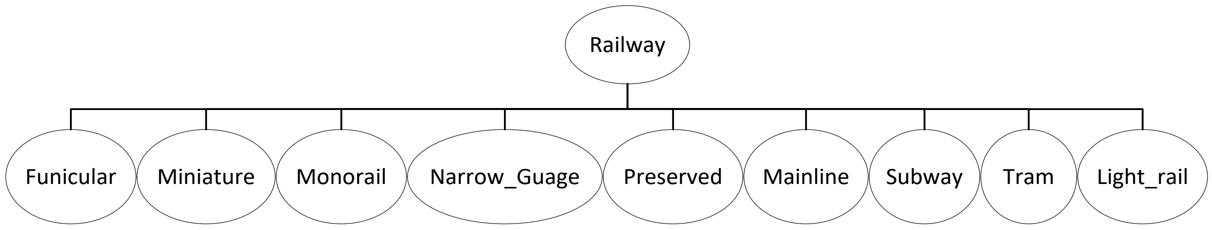


Figure 65 Representation of the node hierarchy for the Railway branch.

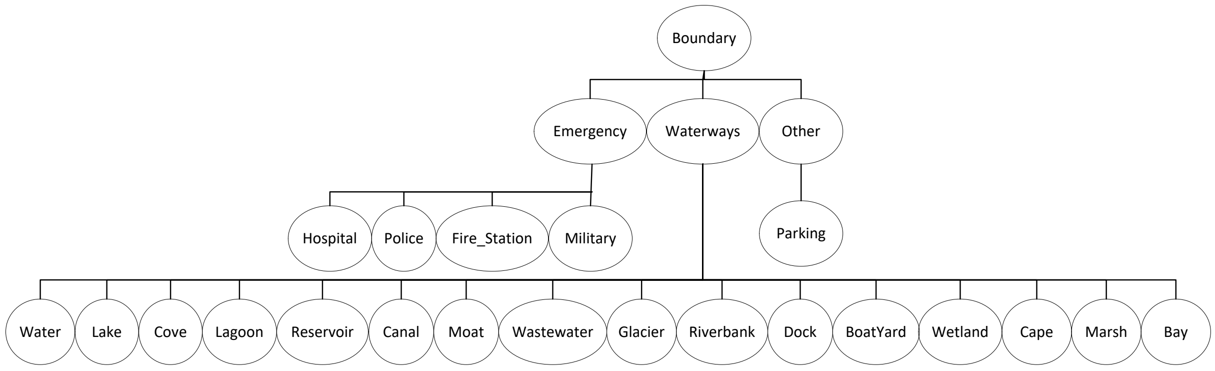


Figure 66 Representation of the node hierarchy for the Boundary branch.

6.5.1 Scenegraph Node

We have stated that the framework utilises the same scenegraph structure, but also stated that the pre-processor scenegraph does not need all functionalities needed by the runtime simulation.

Here we discuss the design for a node of the scenegraph structure. The node is designed to contain a reference to its parent, and a list of children. If a parent is not available, the design should account for this as to minimise potential crashes, and to check while recursively searching up or down a tree.

A node must allow a 3D model mesh object to be attached, or set to null. If the reference is null additional checks should be made when searching for models of particular type; building, highway, stream etc.

The node of the scenegraph must allow many recursive functions which allow the dispersal of data and data-structures. The data and data-structures which are apparent are; Spatial controllers, Material controllers, Effects, Materials, Parent references, and Child references.

Scene nodes of the scenegraph need the functionality to be removed, added, or be reconfigured. Utilising an IVI will provide the functionality to do this, as well as the need for functionalities to reconfigure the parent/child reference relationship.

6.5.2 Spatial Controller

The use of controllers is needed to modify the parameters of the material object stated previously. The same need is apparent for use with the nodes and specifically the spatial parameters of a node; translation, scale, rotation. Controllers are needed, which a node can subscribe to, to allow the updating of its internal data structures and spatial parameters.

6.6 Indexed Array Shader Function Design Structure

The use of an IASF is needed for the benefits it brings to visualisations. An IASF is designed to reduce the use of branches within the GPU processing within the effect. A branch is when an if-statement occurs within the effect, which produces additional overheads, and calculates both sides of the if-statement. An IASF removes these issues by creating separate functions, and determines the function to use on the CPU and not the GPU; thusly removing the need for if-statements. This creates a large number of functions and duplications of program code. The number of rendering techniques and number of lights directly impacts the number of functions which have to be written. This is a small price in development time, which improves rendering speeds and removes processing overheads.

Within computer games, and other applications, a model mesh object will be rendered within a single effect. This effect and the model mesh are tightly coupled, and data passed within the vertex buffer need to match the data to be used within the effect. The data needed for different effects may differ by UV coordinates, tangent and binormal values, which are held within the vertex buffer. If an effect is used on a model without this data, then one of two things may occur; system failure, or the model will not be rendered.

Figure 67 shows the processes of combining multiple effect files into a single referenced file. This means the model mesh objects which will use the IASF effect file will need all data pertinent to all shading techniques. This will increase the overall memory needed for the model mesh, but will allow greater flexibility with rendering large varying assets. This is justifiable within a real-world visualisation system.

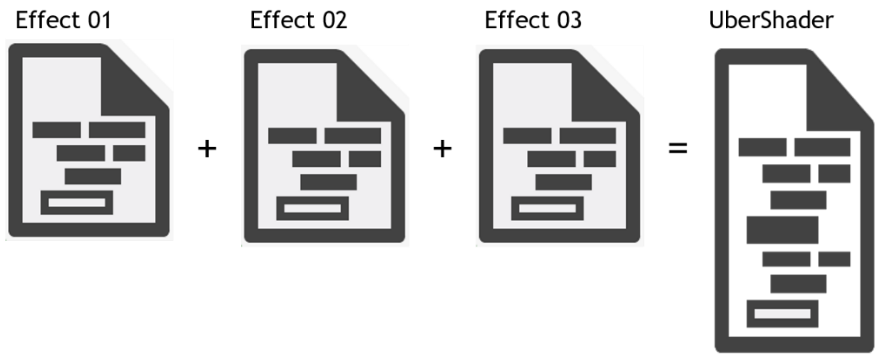


Figure 67 Indexed Array Shader Function Effect File

The IASF will need to combine the following techniques in specific orders, and allow up to 3 lights which are of direction and point style of light to be utilised.

The combination of; per-vertex lighting, per-pixel lighting, Blinn-Phong shading, Phong shading, diffused textured, normal mapped, specular mapped, reflection, refraction, and hemispherical.

Per-vertex is less process intensive than per-pixel, but per-pixel lighting produces higher realism. The use of both allows for the shading model to be dropped from per-pixel to per-vertex to preserve processing. The same justification is made for Blinn-Phong and Phong shading. Diffused texturing, normal mapping, and specular mapping are used to increase realism and effects within the visualisation. These are standard rendering techniques within the computer game industry.

The use of reflection is to be used with water, and the sky box. It is also to be used within the rendering of windows and reflective surfaces. The use of refraction is to be used with objects such as glass for windows of buildings.

Hemispherical rendering is used to apply a blend between two colours of the ground and the sky colour. For example, the reflection of the bottom of the building within a field will be given a green tint, while the top will be given a blue tint which represents the reflection from the sky.

The design of the algorithm will allow users to pick and choose any combination of the individual rendering techniques.

The use of per-vertex and per-pixel is not to be used within the same rendering pass. The same is true for Blinn-Phong and Phong shading.

6.7 GPU Render State Selection Design

Rendering models by the GPU will need to put the GPU into a specific state. The state of the GPU needs to be set for the BlendState, CullState, RasterizerState, and DepthBufferState.

The BlendState has 4 settings; opaque, translucent, additive, and non-premultiplied alpha. The CullState has 3 settings; cull none, cull clockwise, and cull counter clockwise. The RasterizerState can be put into 2 states; solid, or wireframe. The DepthBufferState can be put into 3 states for common rendering techniques; disable depth writing, enable depth writing, and depth test but no writes.

To select the render state for each object to be rendered, there are a number of approaches;

- Approach 1 - Every rendered object for itself.
  - This states that every object to be rendered, sets the GPU to its own rendering state without taking into account the previous state of the GPU. This procedure may be suitable for a small number of objects but for a large number will be exhaustive and waste processing setting the GPU rendering states.
- Approach 2 - Reset all GPU states.
  - After setting the current needed states for the GPU, the object being rendered will reset all states of the GPU regardless of what has and has not changed. This is a very slow approach but allows certain freedoms when creating new objects to be rendered.
- Approach 3 - Objects reset only state which have changed.
  - The object changing the GPU states, will reset only the states that it has changed; theoretically setting the GPU to its default state.
- Approach 4 – Every object changes state only when states are different from previous the object.
  - Each object checks whether any GPU states have changed from the rendering of the previous object.

We have chosen approach 4 for our procedures. The implementation of choosing these is stated in the next chapter.

Assets within the scene can be grouped together depending on the render state which it uses. This is depicted in Figure 68. This reduces the changes to the GPU. The grouping of assets with the same GPU states will minimise the changes in graphics state, improved rendering rates.

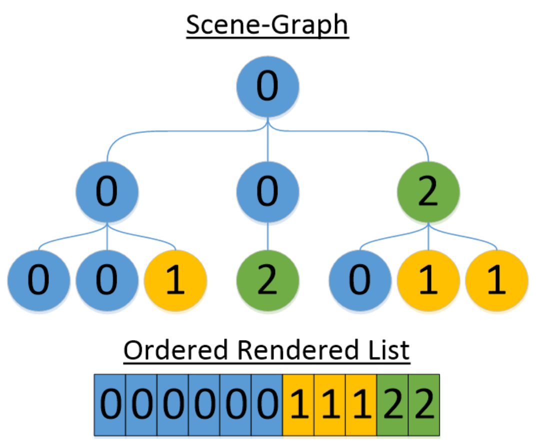


Figure 68 Organising render states of objects for the generation of an ordered array of objects to reduce GPU state changes.

6.8 Improved LiDAR Data Storage Techniques

Textures have been used to store data other than pixel colours. Data is encoded into the binary bits. As stated, the storage needed to store a country's worth of terrain data in ASCII format is vast. We have designed an RGBA texture format and accompanying process which terrain data can be encoded into.

The highest point within the UK is 1344 meters above sea level. The lowest point is 300 meters below sea level. Having a negative number needs a bit from the data structure used for the plus or minus sign. Modifying this value so it cannot be a negative number removes this cost. The value which points can represent will not be between 0 – 1644 meters. With data points of the terrain, as the LiDAR data points we have, are represented with up to 6 points of precision e.g. 7.000001 meters. Only up to 2 points of precision should be encoded giving the meters and centimetres. To improve the data further, the removal of the decimal point from the point value is needed. To remove the decimal point, multiplying the height value by 100 will be sufficient. This will give the value of any data height point to between 0 – 164,400 centimetres. This value can be encoded into 18 binary bits. Within a texture format of 4 bytes, 1 byte for red, 1 for green, 1 for blue, and 1 for alpha, the 18 bits will use the red byte, green byte and 2 of the blue bytes. The highest value for the 18bits is 262143. This leaves 14 bits for use with other encoding parameters.

The process to encode data is shown in Figure 74. The input is the terrain data file, and each data point is parsed and converted to an RGBA pixel data structure. The output of the algorithm will be a texture object which can be used in alternative applications.

```
Input is the terrain map data file; OS or LiDAR
Start
for-each data point in terrain map
        Parse point value from ASCII file
        Add 300 to value
        Round to 2 decimal places
        Multiply value by 100
        Convert value to pixel colour 4 bytes RGBA
        Insert into texture 2D object at same index positon
End
```

Figure 69 Pseudo algorithm for parsing terrain height data to a texture format.

The other 14 bits of data are free to encode other data such as colour, shader techniques, or parameters for shader effect files. For example, the processing of the terrain data files into model mesh objects, data can be decoded to state the colour of the created vertex value. Due to the fact that a number of bits have been already used, the colour values need to be encoded again, i.e. a single bit states that the colour is green, which is used as the vegetation of the ground, a bit can be for the colour blue, used for buildings etc.

Figure 70 shows a representation of the encoding which we have stated here. Next, we discuss the comparison between the already used ASCII format, and the format which we have stated above.

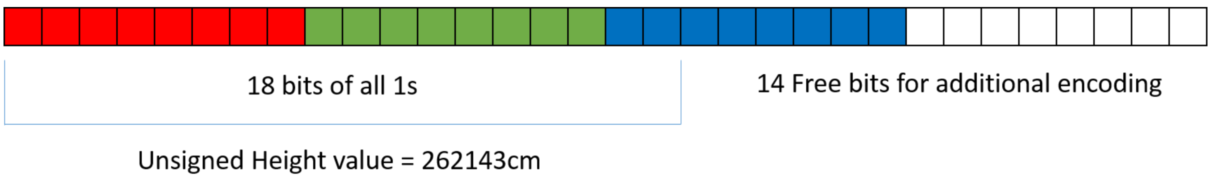

18 bits of all 1s
14 Free bits for additional encoding
Unsigned Height value = 262143cm

Figure 70 Binary encoding of height points within the UK for use within texture objects.

Using the 32 bit, 4 byte method discussed, the data can be encoded into colour pixel values of a texture object. The data can be further improved by removing the 14 bits. This encoding is a common technique for texture formats. We explain the situations on using the 4 byte method as stated.

The LiDAR data is currently stored as ASCII format, as already stated. ASCII writes data as individual bytes for each character of data. Height points written to file can be up to 4 characters to the left of the decimal point, a character for the decimal point and up to 6 characters to the right of the decimal point, and a character of white space separating the height values. This gives 12 possible characters which is 12 possible bytes for each height point written to disk. The maximum storage need for the 25cm resolution 4000^2 points which is used for representing $1km^2$ area coverage would equate to; 4000^2 * 12 bytes = 192 MB. The maximum value for a texture object would be 4000^2 * 4 =

64 MB. This value will be a constant value, whereas the ASCII format is a changing value. This has the potential saving of 66%.

6.9 Interactive Visualisation Interface Design

We have stated the need for an IVI. The IVI will allow access to the algorithms and data structures exposed to the user. Algorithms and data structures such as; scenegraph, PG of queries applied to nodes of the scenegraph, the parameters of the materials attached to the model parts, access to models, the lighting manager, the camera systems, and many helper functions which create and disperse data through the scenegraph. The IVI will also be utilised as a form of debugging for both development, and user interaction and teaching. Details of the components should be provided as well.

Allowing users to add textures, or 3D model mesh objects, the IVI needs to allow access to the user's desktop PC.

The design of the IVI is to combine similar components together within the same overlay window.

The similar components within the main for are; tree view pane, the lights interface, the camera interface, query search pane, and the location tracker. The tree view pane shows the scenegraph of the scene allowing users to click on a node and bring up information on the node. The light's interface gives access to the three lights within the scene. The lights can be modified spatially and visually as well as change the type of the light; directional or spot. Model mesh objects can get access to the scene lights. Changing the values on the interface form, will change the values upon the model mesh who references the lights. The lights form will also give the user to force the light references onto all models within the scene by overriding the locally stored light references held within the material object. The camera interface allows minimal interaction with the camera system allowing change in properties such as field-of-view, position, view direction, and camera type; perspective or orthographic. The search tab allows searching of user generated queries applied to the scenegraph which will return all nodes which satisfy the search result. The location tracker will navigate the virtual camera to a location specified in latitude and longitude coordinates, as well as X and Y Cartesian coordinates.

From the main overlay form, the access to sub-components, or components which need large screen coverage.

The sub-components are; node form and the material form. The node form gives access to the spatial properties of the node as well as the allow access to the model mesh data. The generation of the spatial and material controllers which is provided on this form. As well as the controllers applied to the node, the setting of the graphics state and IASF is provided. A separate form is used for the material of each of the model mesh parts which gives access to all HLSL properties.

The model-part material form is only accessible from the node form, which in turn is only available from the main form.

The use of sliders, check boxes, buttons, combo-boxes and many other features will be needed. Sliders will provide increased speed with changing variable values.

Figure 71 shows the design flow between the components. A number of screens will be available. The tabs of the screens will allow similar components to be grouped, or easily navigated.

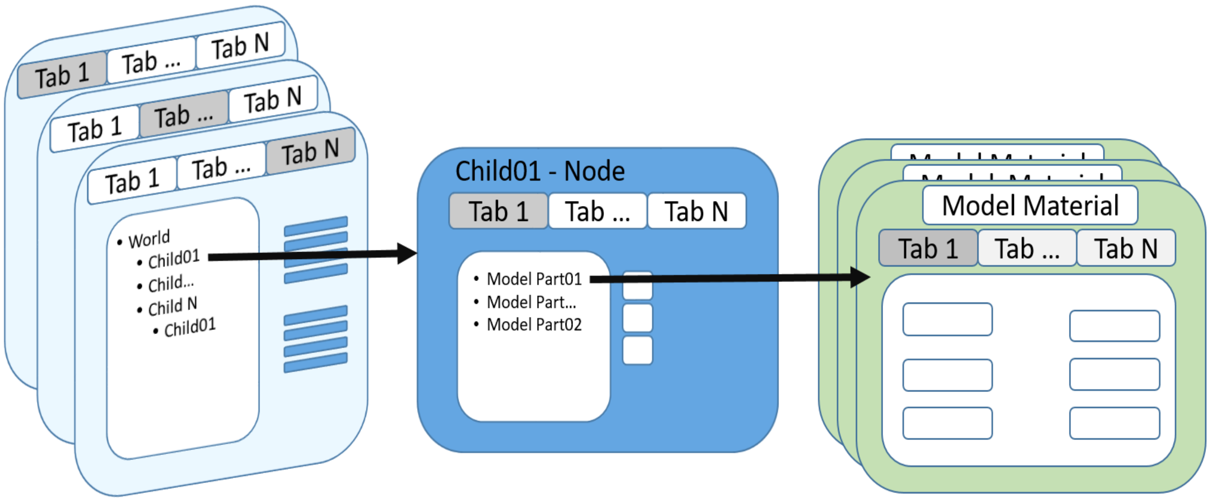


Figure 71 Interactive Visualisation Interface Design.

6.10 Conclusion

Within this section we covered the high-level design of the framework. We have stated the external libraries, and internal libraries which are needed. We have designed the pre-processes which are needed, along with the runtime processes.

The algorithms which are to be used within the pre-processor have also been shown. These algorithms involve the processing of terrain map data, and the geospatial data of OSM for the PG of data and 3D model mesh assets.

A novel scenegraph structure is designed to be used in the pre- processes and the runtime processes. Slight differences are present between the pre-processor and runtime scenegraph structures. Within the nodes of the scenegraph a custom model object can be attached. The model contains model parts which also contain the material object. The material object is used to communicate and pass data to the effect file. The effect file is used to encode a large number of rendering techniques into a single effect file which is called an IASF. The use of an IASF has not been applied within the domain of GIS and geospatial visualisation. The designed IASF allows for automated shader selection to be done during simulations with the monitoring of simulation parameters.

The designed IVI will allow user interactions with algorithms, and components of the framework.

The implementation of the designed algorithms will provide a framework which allows from start to finish a single, customisable pipeline for the visualisation of real-world urban environments using big geospatial data.

Implementing the algorithms and data-structures is presented in the next chapter.

7 PVS Pre-Processor

This chapter presents the external API's utilised within the PVS framework and their impact on pre-processing algorithms. The pre-processing algorithms used to generate real-world virtual scenes of big geospatial data sets is also presented.

7.1 Software

We utilise the Microsoft XNA[43] rendering pipeline. Although Microsoft has stopped maintaining the API, it is still relevant for this work and research. The XNA API is to be chosen for its pre-processing capabilities. It allows assets commonly found within computer games and visualisation software to be compiled into a purpose built binary format which is then loaded at runtime.

Visual Studio 2013 Professional is used as the IDE. The use of Visual Studio allows the input of parameters at compile time through the IDE itself. These parameters can adjust, or modify the internal processes of the pre-processor.

The hardware for which the framework is developed on is a mid-range laptop - a 64bit Windows10 Operating System, 8GB of RAM, Intel Core i7-3630QM 4 (2 logical cores per physical) core CPU at 2.40GHz, and an Intel HD Graphics 4000 Chip with an additional NVIDIA GeForce GTX 660M. The hard-drive of the computer is a Solid State Drive (SSD). An SSD has faster read and write properties than HDD.

We discuss further in the next section the intricacies of the Microsoft XNA API.

7.2 MS Build – Compiling Assets

MS Build is needed for the compilation of large number of assets, and the PG of model meshes. The pre-runtime compilation is needed for due to the time it takes to compile assets.

The use of MS Build allows the compilation of assets for games applications and Abstract Data Type (ADT). This ability is a major factor why Microsoft XNA[44] API is still a viable option for professional development and academic research.

The MS Build[45] process allows for a single pipeline to compile and optimise XNB[46] files format to be read by default content importers, or by custom content readers at runtime.

[43] https://www.microsoft.com/en-gb/download/details.aspx?id=23714
[44] https://msdn.microsoft.com/en-us/library/bb203940.aspx
[45] https://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx
[46] http://xbox.create.msdn.com/en-US/sample/xnb_format

The process incurs the following functions; import, process, write, and read. The import function imports objects into the processor. The processor either converts the object to another ADT, or pass the object to the writer. The writer will write the ADT to a file. A custom binary writer is needed for generated ADTs. The role of the reader is to read in the data at runtime. A binary reader is needed to read the ADT into a runtime class object.

7.3 External Libraries

Within this section we state the external libraries chosen to utilise. The libraries chosen are open source and free from restricting licences. The libraries consist of an animation and interpolation library, triangulation library, a dynamic LINQ query library, and DotNetCoords.

7.3.1 Animation / Interpolation library

The animation/interpolation library is called 'Tweener'[47]. This library is open source which allows the extension methods of Catmull-Rom interpolation, which is needed for the PG of terrain points and highway structures. The library also allows interpolations of; linear, quadratic, cubic, quartic, quintic, sinusoidal, exponential, circular, elastic, back, and bounce.

Within the pre-processors, it is used for the PG of additional data points, e.g. Vector2 or Vector3 positions for terrain and highway interpolations. Within the runtime simulation it is to be used to animate the spatial properties of nodes, and the visual properties of a models material properties.

The library works dominantly by using C# delegates[48] . The constructor of the Tweener object takes in the *start* value, the *end* value, the *duration time* it takes to interpolated from the *start* value to the *end* value, and the function to use (the interpolation functions we stated previously).

The library allows basic animations; start to finish, and looping from start to finish. The latter repeats the animation when it reaches the end value. We have extended the library to allow for additional looping features; repeat in reverse. This feature allows the backwards and forwards animation, creating a pulsating effect. Using these three adapted functions, a large amount of effects can be achieved.

The library also contains functions for easing-in, easing-out, and ease-in/out. We will utilise the latter. The easing methods will smooth out the beginning or ending of the interpolation.

Each function have four parameters passed in;

- Elapsed time - the total elapsed time of the current simulation.

[47] https://xnatweener.codeplex.com/
[48] https://msdn.microsoft.com/en-us/library/ms173171.aspx?f=255&MSPPError=-2147217396

- Start value – self-explanatory.
- Change value - value is calculated using the total elapsed time and a time value from the previous frames elapsed time.
- Duration value - how long the interpolation will take.

The class diagram is shown in Figure 72.

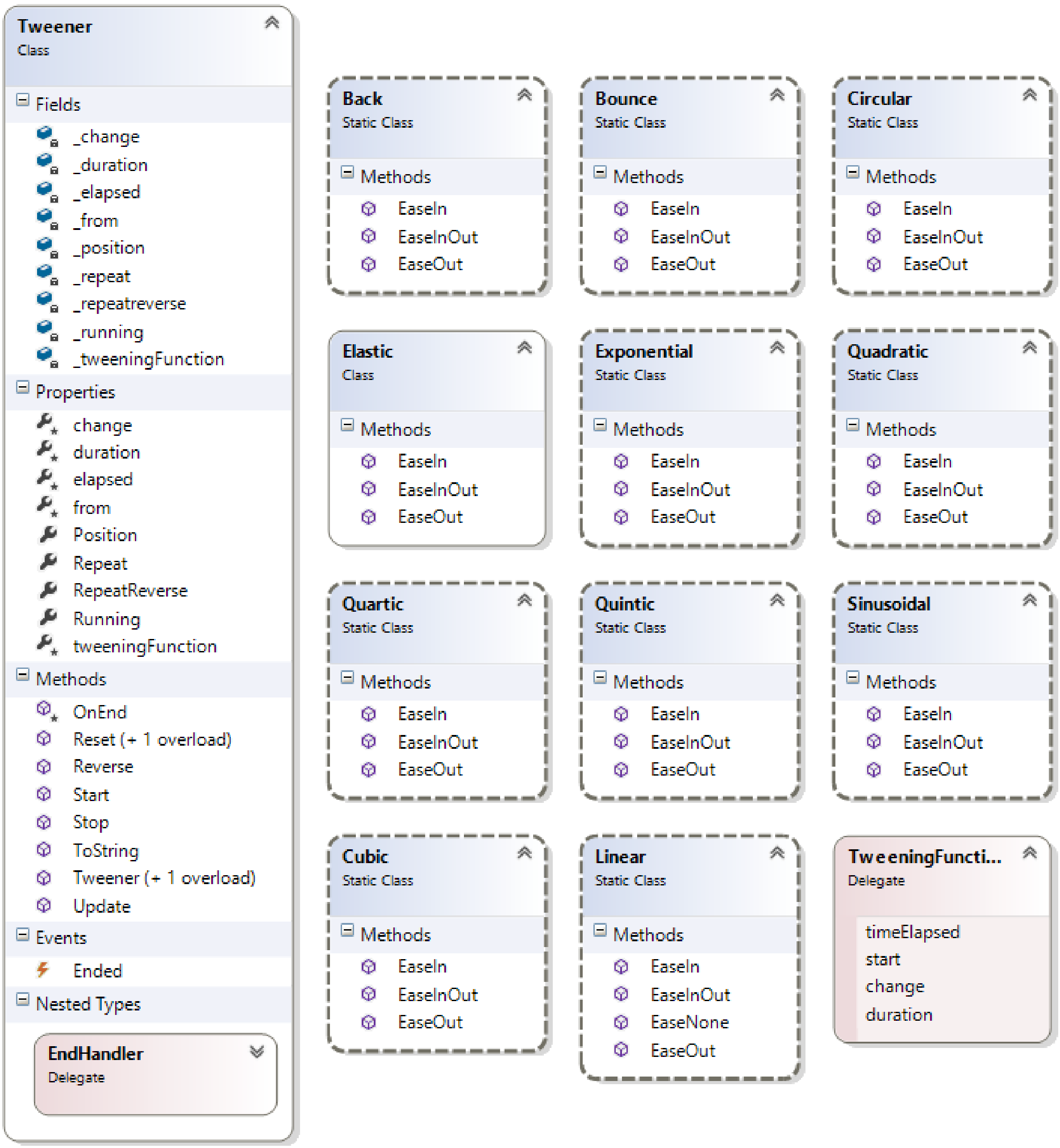


Figure 72 TweenerLibrary Class Diagram

7.3.2 Triangulation Library

The Triangulation[49] library is used to create triangular vertex and index arrays to be generated from an arbitrary array of 2D points. This includes cut-outs of the 2D polygons; needed to create rooftop structures of real-world buildings. The triangulation algorithm is an implementation of David Eberly's ear clipping algorithms presented in [79]. The implementation allows the inputting of 2D vertices and returns a triangulated 2D polygon from the array of 2D vertices in the same order, as well as the indices needed for optimisations in the rendering pipeline. Other triangulation algorithms which are free to use such as the Triangle.Net[50] library does not provide this heavily needed utility.

We have extended the source code with additional overloaded functions which will convert inputted properties of varying datatypes used within the XNA library.

A limitation to the algorithms is if the polygonal structure has overlapping lines sourced from the OSM data; e.g. a hole in a polygon overlaps the outside polygonal structure. We will not check this issue programmatically for the prototype, but issue that it may cause visual errors within the rendering at simulation time.

This library satisfies the needs of the specification for the creation of procedural model mesh assets of real-world objects; buildings, highways, boundaries etc.

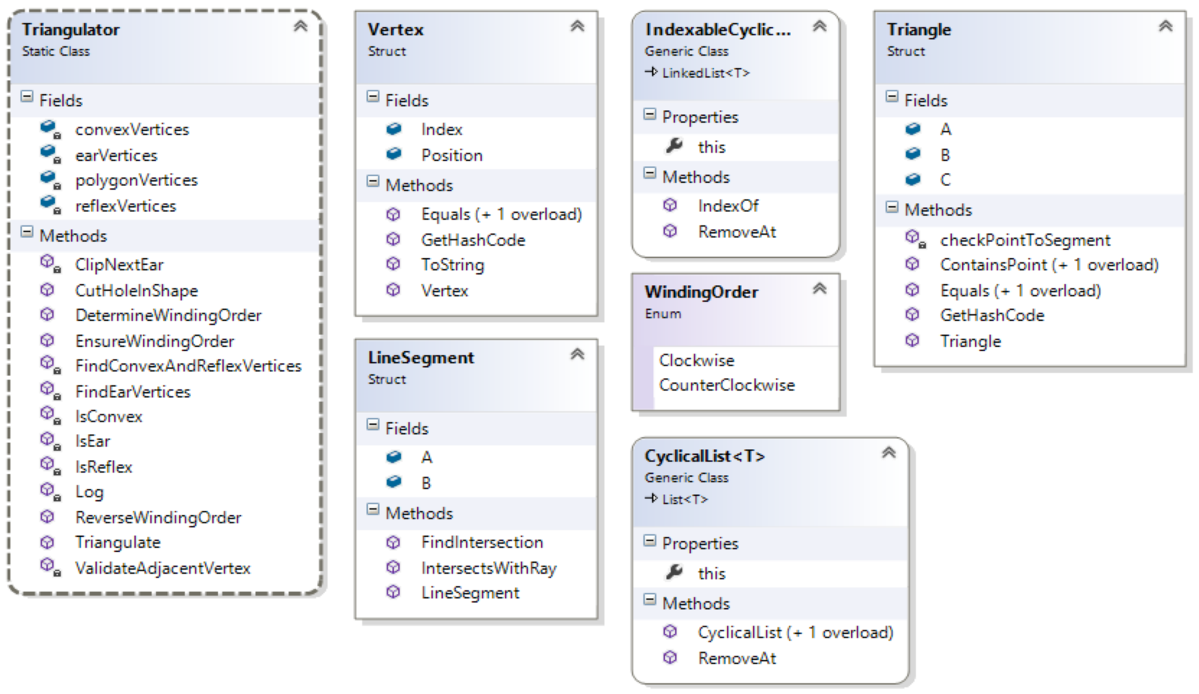


Figure 73 Triangulation Library Class Diagram

[49] https://triangulator.codeplex.com/
[50] https://triangle.codeplex.com/

7.3.3 System.Linq.Dynamic

The System.Linq.Dynamic[51] library is an extension to the .Net LINQ library. It is utilised for the creation of user generated queries. The library allows for the dynamic creation of string queries which are applied to types of object within the framework. This allows a user to query any number of permutations of parameter checks of objects within the framework and .Net library. The library exposes the parameters of the object using reflection. Searching the scenegraph for a node with the name of 'Liverpool Liver Building' can be undertaken. The string representation needed for this would be *PVSNode.name == "Liverpool Liver Building"*; the class object, then the property of the class, and then the expression. Multiple expressions can be queried; *'PVSTagBuilding.Fireproof == true and PVSTagBuilding.Height > 10'*. This example checks if the tag of the node which is a building has a property called Fireproof, and Height, and checks if true and over 10 respectively.

Combining the expression library with the novel scenegraph structure allows searching complete scenes for user queried assets.

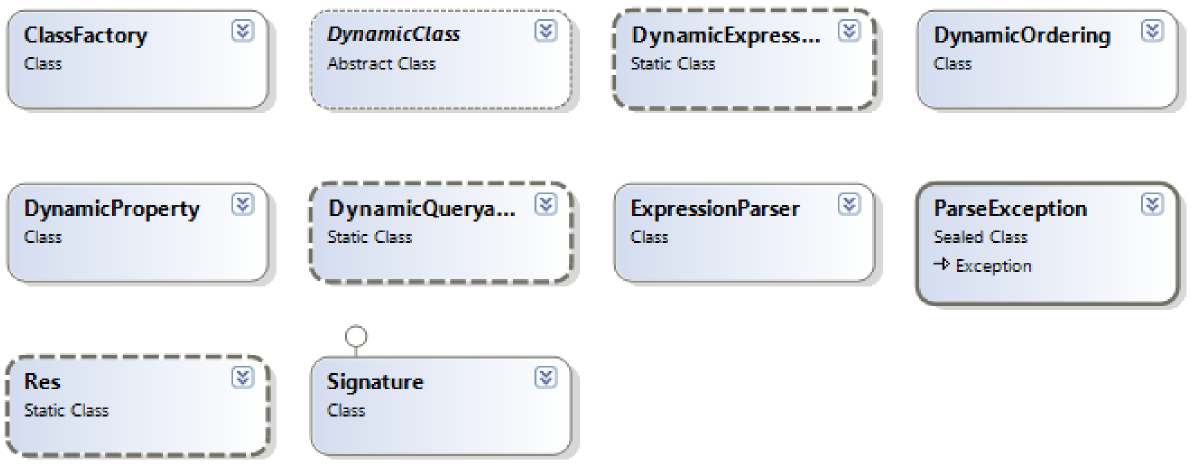


Figure 74 System.Linq.Dynamic Class Diagram

7.3.4 DotNetCoords

The DotNetCoors[52] library is used to convert from one mapping coordinate system to another. This will be used to map from Latitude and Longitude to X, Y, Z coordinates used by the XNA API.

7.4 PVS Camera Library

The Camera Library satisfies the specification stated. It contains many 3D camera systems, and 2D camera systems with the addition of extra 2D cameras than stated within the specification needs.

[51] https://www.nuget.org/packages/System.Linq.Dynamic/
[52] https://www.doogal.co.uk/dotnetcoords.php

The library contains the following 3D camera systems which can be employed within the simulation; Arc Ball camera, Chase camera, First-Person camera, Free roaming camera, Target camera.

The specification section covers the characteristics of the individual cameras.

The 2D cameras contained are;

- 2DCameraBasic - this camera allows the viewing and projection of 2D elements which are always facing the screen. The projection of the 2D camera can be translation, and scaled, but not rotated.
- 2DCameraParallax - this camera is attached to a 2D object within the world. If the object moves then the camera will follow. The Super Mario games and earlier 2D platformer games use this type of camera. The effect is that the object stays within the centre of the screen and the world translates around the object.
- 2DCameraYawPitchRoll - this camera allows the same functionalities as that of the '2DCameraBasic' and the addition of rotation.

Each of the cameras stated has an update method to update its internal parameters.

Figure 75 shows the inheritance between the camera class objects and their interfaces. The interfaces consist of;

- ***IPVSView*** – a helper interface which states that complying classes need to implement a method which returns the 'view' matrix of the camera.
- ***IPVSCamera*** – This interface combines the '***IPVSView***' interface to save additional coding. It is also the base for all 3D cameras and holds data such as; the GPU reference, the view matrix, projection matrix, the bounding frustum of the camera, the field of view, near plane, far plane, as well as the aspect ratio of the screen being rendered too. The interface updates the camera settings, and creates the perspective view matrix from the field of view, aspect ratio, near plane, far plane, and the projection matrix. It also states that complying classes need to implement specific functions such as returning the current position and direction of the camera. These are critical to the rendering of objects within the rendering pipeline.
- ***IPVSCamRotatable*** – this interface states that complying classes need to implement a function to allow the rotating of the camera and update the relevant view and projection matrices appropriately.
- ***IPVSCamMovable*** – this interface states that complying classes need to implement a function to allow the translation of the camera and update the relevant view and projection matrices appropriately.

- ***IPVSCamMoveRotate*** – this is an interface which combines the '***IPVSCamRotatable***' and '***IPVSCameMovable***'.

Using the interfaces, the large number of varying cameras needed for a modern GIS can be achieved. The implementation satisfies the needs set out within the specification.

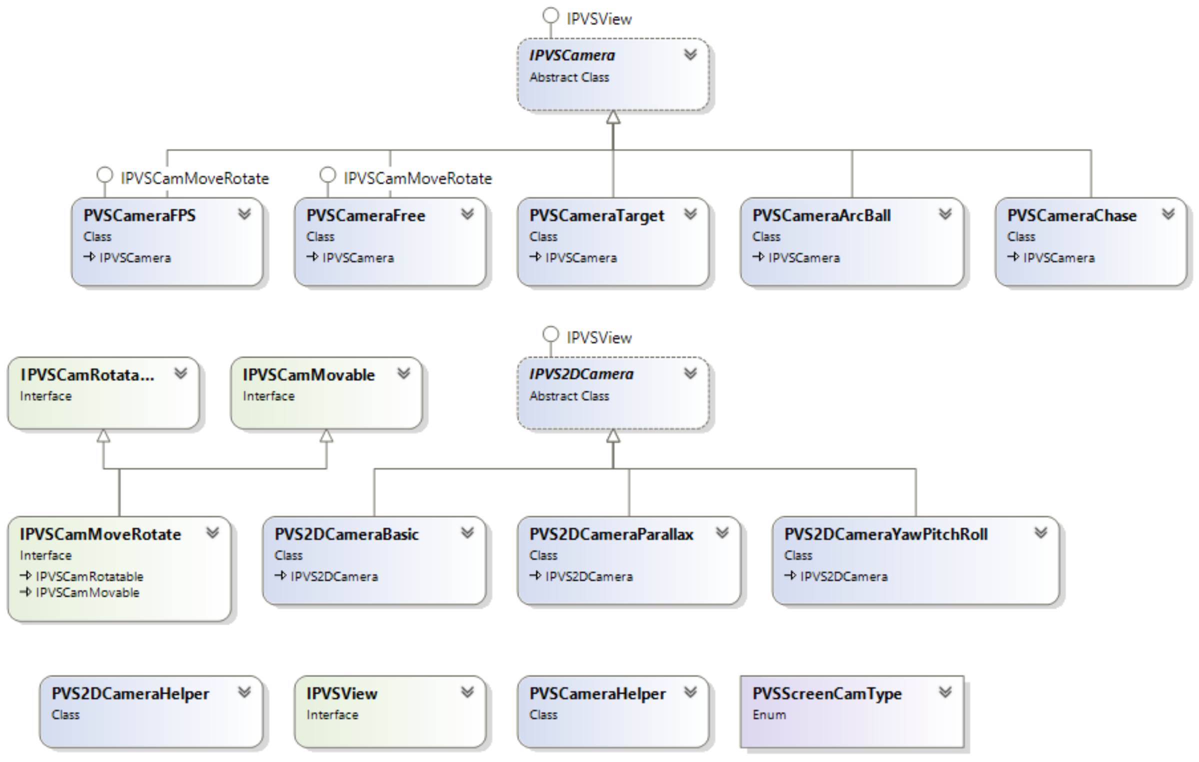


Figure 75 Camera Library Class Diagrams.

7.4.1 Camera Helper Classes

Within the library, the interfacing between application/simulation and the cameras, a helper class is used. These helper classes employ static properties, and static methods. Each property or method is used as a single point for updating multiple cameras easily, or receive data from the cameras without needing to know the object types.

The 3D camera helper utilises static methods to update the cameras regardless of the type of camera it is; Arc Ball, Free, etc. Using the .Net and C# Generics this can be achieved. The helper functions can be applied by varying inputs (keyboard, mouse, gamepad) through the inputted reference to the 'Input Manager'. The basic functions check whether certain buttons or keys have been pressed within the last frame and adjusts the camera accordingly. This can be translating the cameras, rotating, or a combination of both.

The helper class also contains constant variables which limit the rotation, and movement speed applied to the camera being updated. Figure 76 show the UML of the camera class helper.

The helper class has static properties of ‘IsCameraDirty’ along with the current ‘EyePosition’ and ‘EyeVector’. The static values are updated when the camera is modified. The IsCameraDirty flag is used in conjunction with the spatial and material controller. The controller will have a reference to the camera helper class, and monitors the state of the ‘IsCameraDirty’. If true, the controllers will force an update to the WorldViewProjection matrix used for placing objects in 3D space at rendering time. The EyeVector, and EyePosition is also used for rendering.

This technique removes the need for multi object passing from the camera system to the material system for rendering. Although initial initialisation is needed to hook the camera system and the material system together through the use of the controllers, but if the controller is removed, the link is severed. This proves beneficial to improve the latency between checks and optimises calls between material and camera system and only updates parameters when they need to be updated.

Figure 77 shows the connection between the static properties of the 3D camera helper and how it links between the material controllers used by each material.

A 2D camera system and accompanying helper class is available. The helper class extends the functionality and provides access to the mouse position in screen coordinates, as well as a pointing direction derived from the mouse position and the view direction. This leads to ray casting within a virtual scene for object picking.

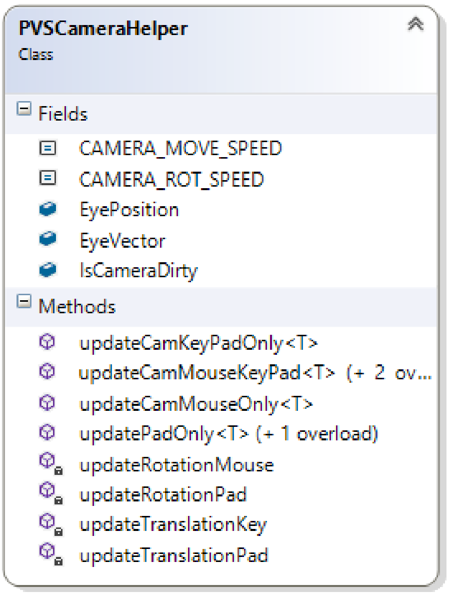


Figure 76 The UML of the implemented camera helper class

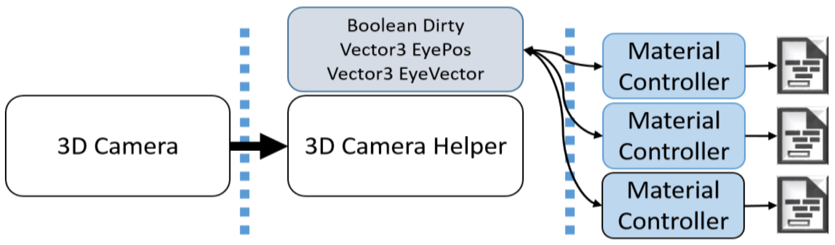


Figure 77 Camera and Dirty flag material hook-up.

7.5 PVS Input Manager Library

The Input Manager Library is made of 2 class objects; the main input manager, and a helper class.

The input manager contains references to; current keyboard state, current gamepad state, and current mouse state. Keeping reference to the previous keyboard state, gamepad state, and mouse state is also tracked. Single keypresses can be tracked, and keys which are held down.

The keyboard helper class allows for adding of additional key char codes so it can be adapted for use around the globe using additional languages.

The library limits the number of controllers and keyboards to inputs to 4. This means that the framework can have multiple users using the simulation at once. This quality satisfies an aspect of the specification. We stated that a modern GIS needs to accommodate multiple users for cross collaboration.

Figure 78 shows the class diagrams of the PVSInputManager class library.

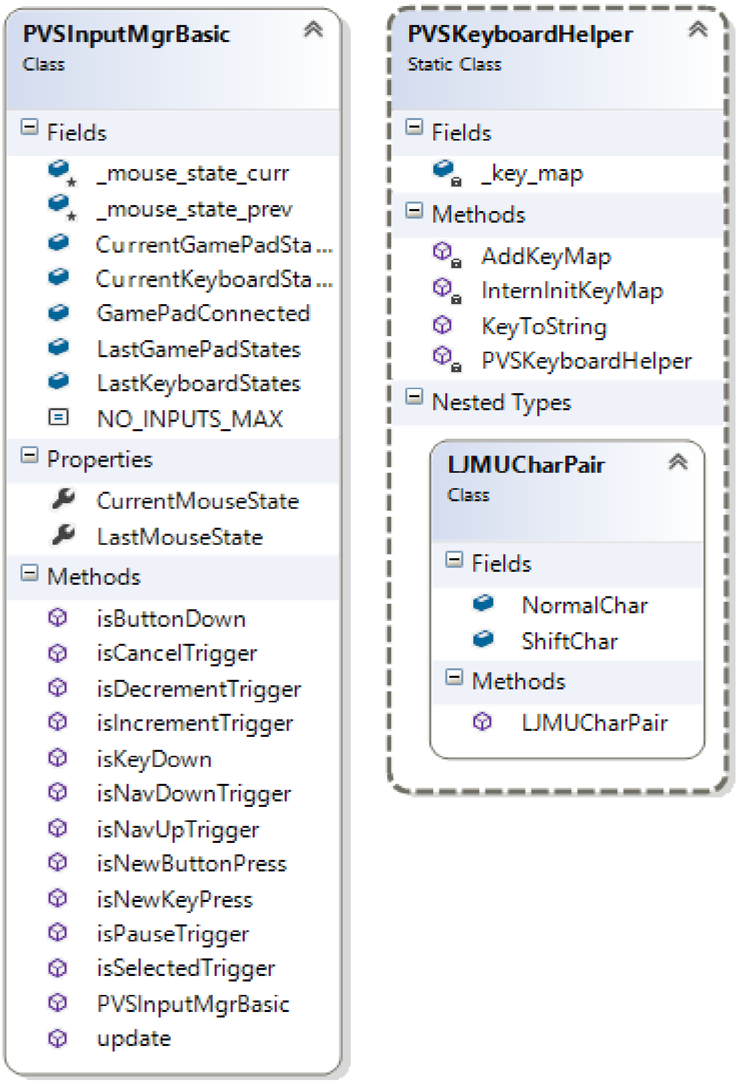


Figure 78 PVSInput Manager Class Diagrams.

7.6 PVS XML Serialiser and Deep Copy Library

The XML Helper library has been created as its own library to improve flexibility and code reuse within the project and external projects. The function of *Serialise* and *Deserialise* utilise .Net generics[53]. Generics allow any object type to be serialised and de-serialised to XML and other schema defined output files. The files will include all references and properties of the object.

The library also contains a function to *DeepCopy* an object in memory. The function creates a binary stream in memory, and it is from this stream a new object containing a binary copy of all parameter values and references is created.

[53] https://msdn.microsoft.com/en-gb/library/512aeb7t.aspx

Alternatives to this is the *MemberwiseClone*[54] function which is classed as a *ShallowCopy*. This will not create a copy of the referenced objects, thusly the copied object will contain references to external objects, creating multiple references to external objects.

The library concludes the need specified within the specification chapter to allow the copying and generation of assets which are the same, but allow separate modifications. The library also allows the serialisation of runtime objects to external files which is needed to bypass the limitations of MS Build which only allows single input and single output.

The depiction between the shallow copy and deep copy is show in Figure 79.

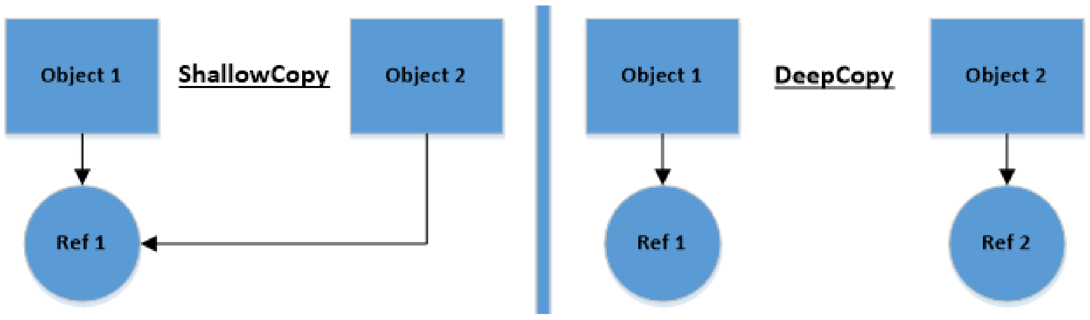


Figure 79 Shallow and Deep copy outputs.

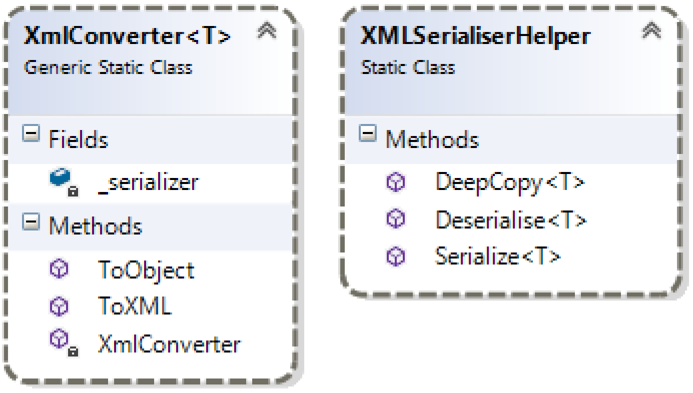


Figure 80 PVSXML Serialiser Helper Class Diagram.

7.7 OSM Enumeration Library

Given the large number of key words held in the OSM data-base, we have generated a comparison library filled with C# enumerations of these keywords which are used throughout the framework. The enumeration values have been put into its own library because of the cross dependence between the pre-processor and the runtime system. The library contains the categorisations of keywords; buildings, highways, sidewalk types, types of barriers etc.

[54] https://msdn.microsoft.com/en-us/library/system.object.memberwiseclone(v=vs.110).aspx

Due to the nature of working with real-world names for objects, some characters of the alphabet are used as the first character of a word which may be illegal within the .Net framework or C# language. Words as this can be "*_lock*", or "*concrete: plates*". For this we use the C# '*DescriptionAttribute'* mechanism[55]. This allows substitution of the illegal words for legal words. For example, if we use the term "lock", the enumeration will return the alternative attribute of "_lock".

We also use enumerations for coding standards. Due to the large amount of variations of keywords within the OSM data-base, errors may occur when developing code to search for a certain type of object. Using the enumerations, spelling mistakes like this can be kept to a minimum.

Another reason for utilising enumerations is the ability to save only the index of the enumeration during the pre-processor writing of assets. A model object of a certain type has the enumeration converted to an unsigned integer and this is saved as binary. As long as the runtime enumeration layout structure is the same, it will assign the same enumeration value to the object. Being that we have put the enumerations within one library which is used within the pre-processor and runtime library, this should not be an issue.

The full list of currently implemented enumerations used as shortcuts and lookups are listed in Table 13. The generated class diagram is too large to fit into this document.

[55] https://msdn.microsoft.com/en-us/library/system.componentmodel.descriptionattribute.description(v=vs.110).aspx

NodeNames	OSM_K_VALUES	OSMTagWaterway
OSMTagPublicTransport	OSMTagBarrier	OSMTagFootway
OSMTagSidewalk	OSMTagAerialway	OSMTagAeroway
OSMTagCycleway	OSMTagLeisure	OSMTagRailway
OSMTagCondition	OSMTagJunction	OSMTagLanduse
OSMTagMilitary	OSMTagCraft	OSMTagPlace
OSMTagPlacement	OSMTagNatural	OSMTagPower
OSMTagRoute	OSMTagShop	OSMTagReligion
OSMTagSurface	OSMTagTourism	OSMTagTrackType
OSMTagSport	OSMTagHistoric	OSMTagOffice
PVSTagBuildingType	OSMTagRelationTagValue	OSMTagTurnLanes
OSMTagTurnLanes	PVSTagAmenityType	OSMTagRoofOrientation
OSMTagGeological	OSMTagMan_Made	PVSTagBuildingCladding
OSMRelationTagKey	OSMTagEmergency	PVSTagHighwaySidewalk
OSMTagRoofShape	OSMTagSmoothness	PVSTagBuildingMaterial
OSMTagRoofMaterial	PVSTagHighwayType	PVSTagBoundaryType

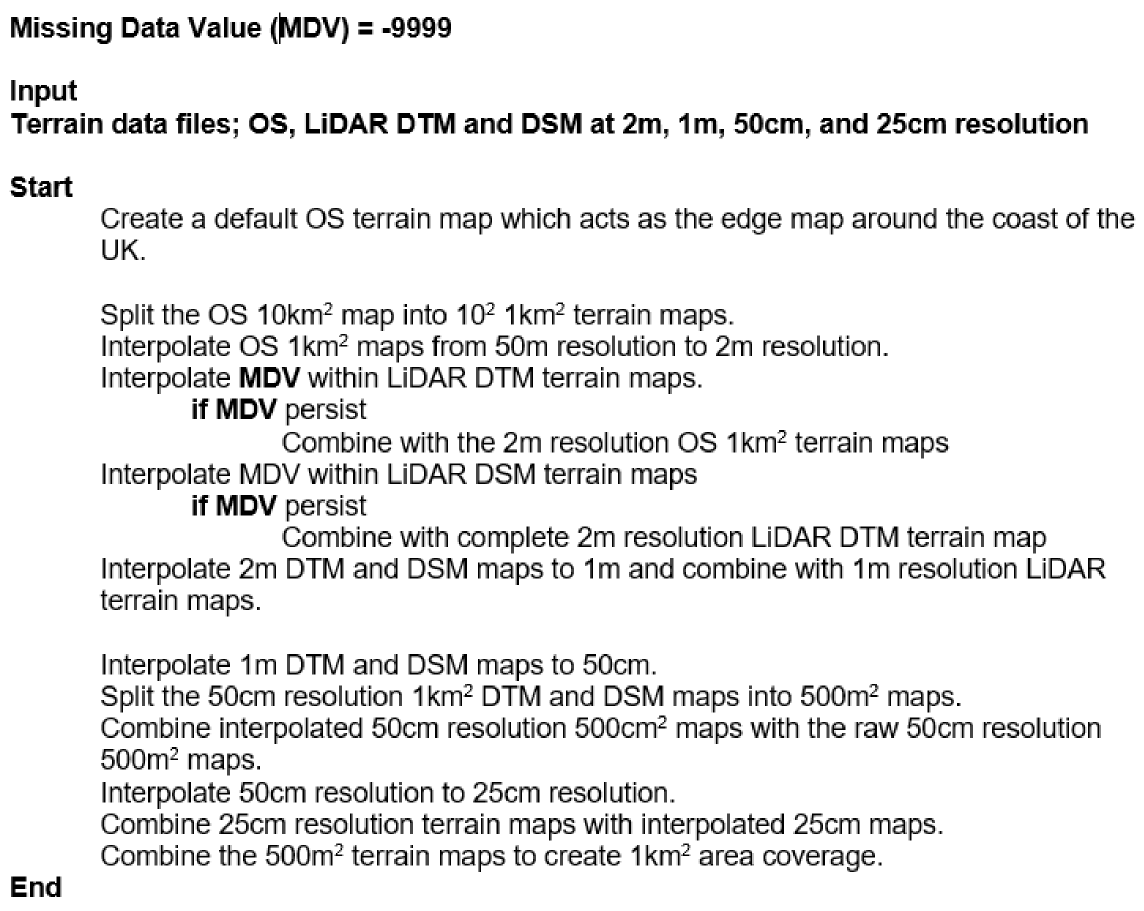
Table 13 Enumeration library constructs.

7.8 Terrain Map Generation Algorithms

OS terrain data is not of high resolution. The data removes large areas of detail through the low resolution. It is however a complete dataset covering the entire UK at 50m resolution. Complete in this essence means that the maps do not contain missing data values. As stated throughout this work, errors of height values may still persist. This is justifiable as a base for interpolation and combining with erroneous LiDAR data sets. LiDAR suffers from missing data, error prone results, and areas which are mapped but at varying resolutions; 2m, 1m, 50cm, and 25cm.

We discuss the generation of the algorithm used to combine OS and LiDAR DTM, DSM data-sets within an external program to be included within the pre-processing pipeline. We utilise a separate program.

The algorithm is stated in Figure 81. The input for the algorithm are the OS, and LiDAR DTM, DSM terrain map files. The output will be complete data sets at each resolution for the LiDAR DTM and DSM and OS 1km^2 maps at 500^2 point resolution.



```
Missing Data Value (MDV) = -9999

Input
Terrain data files; OS, LiDAR DTM and DSM at 2m, 1m, 50cm, and 25cm resolution

Start
    Create a default OS terrain map which acts as the edge map around the coast of the
    UK.

    Split the OS 10km$^2$ map into 10$^2$ 1km$^2$ terrain maps.
    Interpolate OS 1km$^2$ maps from 50m resolution to 2m resolution.
    Interpolate MDV within LiDAR DTM terrain maps.
        if MDV persist
            Combine with the 2m resolution OS 1km$^2$ terrain maps
    Interpolate MDV within LiDAR DSM terrain maps
        if MDV persist
            Combine with complete 2m resolution LiDAR DTM terrain map
    Interpolate 2m DTM and DSM maps to 1m and combine with 1m resolution LiDAR
    terrain maps.

    Interpolate 1m DTM and DSM maps to 50cm.
    Split the 50cm resolution 1km$^2$ DTM and DSM maps into 500m$^2$ maps.
    Combine interpolated 50cm resolution 500cm$^2$ maps with the raw 50cm resolution
    500m$^2$ maps.
    Interpolate 50cm resolution to 25cm resolution.
    Combine 25cm resolution terrain maps with interpolated 25cm maps.
    Combine the 500m$^2$ terrain maps to create 1km$^2$ area coverage.
End
```

Figure 81 *Pseudo algorithm to combine and interpolation terrain maps of OS, LiDAR DTM and DSM at 2m, 1m, 50cm, 25cm resolutions.*

Within the algorithm, a default map is generated by analysing the LiDAR terrain maps at the coastal line of the UK. The default value for all points within this map is -2.7 meters. This map is also used to extract tangent values which are edge lines of maps, used within the Catmull-Rom interpolation function. Figure 82 shows the default maps needed for the UK in green.

With complete maps, data can be inferred. This is especially pertinent by using the DSM maps to gather information for height of buildings on the terrain. Using this process will also guarantee a country's worth of terrain data is available in all 3 formats; OS, LiDAR DTM, and DSM. Data storage techniques for OS, LiDAR DTM and DSM is needed.

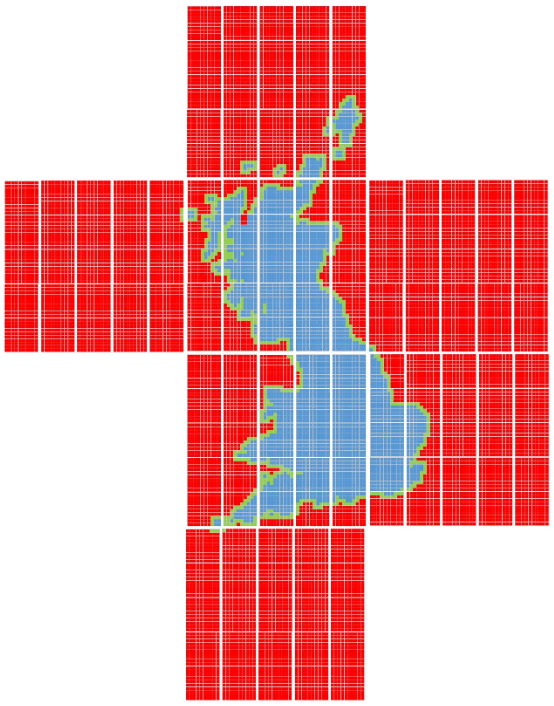


Figure 82 British Isles and the default edge maps in green.

7.9 PVS Pre-processor

Due to the size of the subsystem we will split this section into separate parts consisting of the high-level overview of the pre-processor with its components and connections to external libraries. This will include a flow diagram of the input data, and the outputs of the processor to create the scenes. Extracting the OSM data structures and information. The PG of the model mesh objects for buildings, highways, amenities, boundaries, and terrain. LiDAR and OS terrain combination for improved terrain model mesh data. Parsing external 3D models converting them to the designed format.

7.9.1 High-level overview of the Pre-processor

This will include the flow of data within the processor which includes the importing and processing of external assets, as well as serialisation of procedurally generated scenegraph nodes to be reprocessed in an external processor to convert them from XML to XNB. The reason for the additional process is that XNA is a single-input, single-output pipeline. Thusly, the serialisation of assets to XML, and then reprocessing the XML files can alleviate this problem.

The algorithm shown in Figure 83 is the high-level process of the pre-processor algorithm to generate scenes. We discuss the steps of the algorithm in the following sections.

The algorithm is also shown in Figure 84 as an activity diagram.

```
Input
Text file with user flags used to select processes.
OSM XML file.
Terrain data
User generated content; diffuse textures, model mesh objects

Start
        Import PVSDataToLoad
        Create OS References
        Import 3D model mesh list
        Set Visual Studio input parameters
        Import OSM XML to class objects
        Convert OSMNodes to Dictionary<TKey, TValue> data structure
        Convert OSMWays to Dictionary<TKey, TValue> data structure
        Create Bounding Box from OSM Bounds
        Generate partition boxes from OSM Bounds
        Generate scene-graph from OSM Bounds
        Generate terrain model meshes from scene-graph bounds
        Loop over OSMWays
                Generate Boundary model mesh objects
                        Check OSMWay ID against user generated content on disk.
                Generate Building model mesh objects
                        Check OSMWay ID against user generated content on disk.
                Generated Highway model mesh objects
                        Check OSMWay ID against user generated content on disk.
                Generate Single model mesh objects
                        Check OSMWay ID against user generated content on disk.
                Generate Waterway model mesh objects
                        Check OSMWay ID against user generated content on disk.
        Assign name if name is Null in scene-graph node
        Assign node path name
        Recursively generate bounds of models
                Recursively advance to leaf node
                        Generate bounds for node
                        Return bounds
                Combine current nodes bounds with children bounds
        Reference parent and child member properties
        Serialise scene-graph nodes to XML
        Write PVSDataToLoad as XNB format.
        Import XML files and parse XML to object
        Write the objects to XNB format.
End

Output
Serialised XML files for each node of the scenegraph.
Formatted XNB files for each node of the scenegraph to be used during runtime.
```

Figure 83 Pseudo algorithm for high-level overview of the pre-processor.

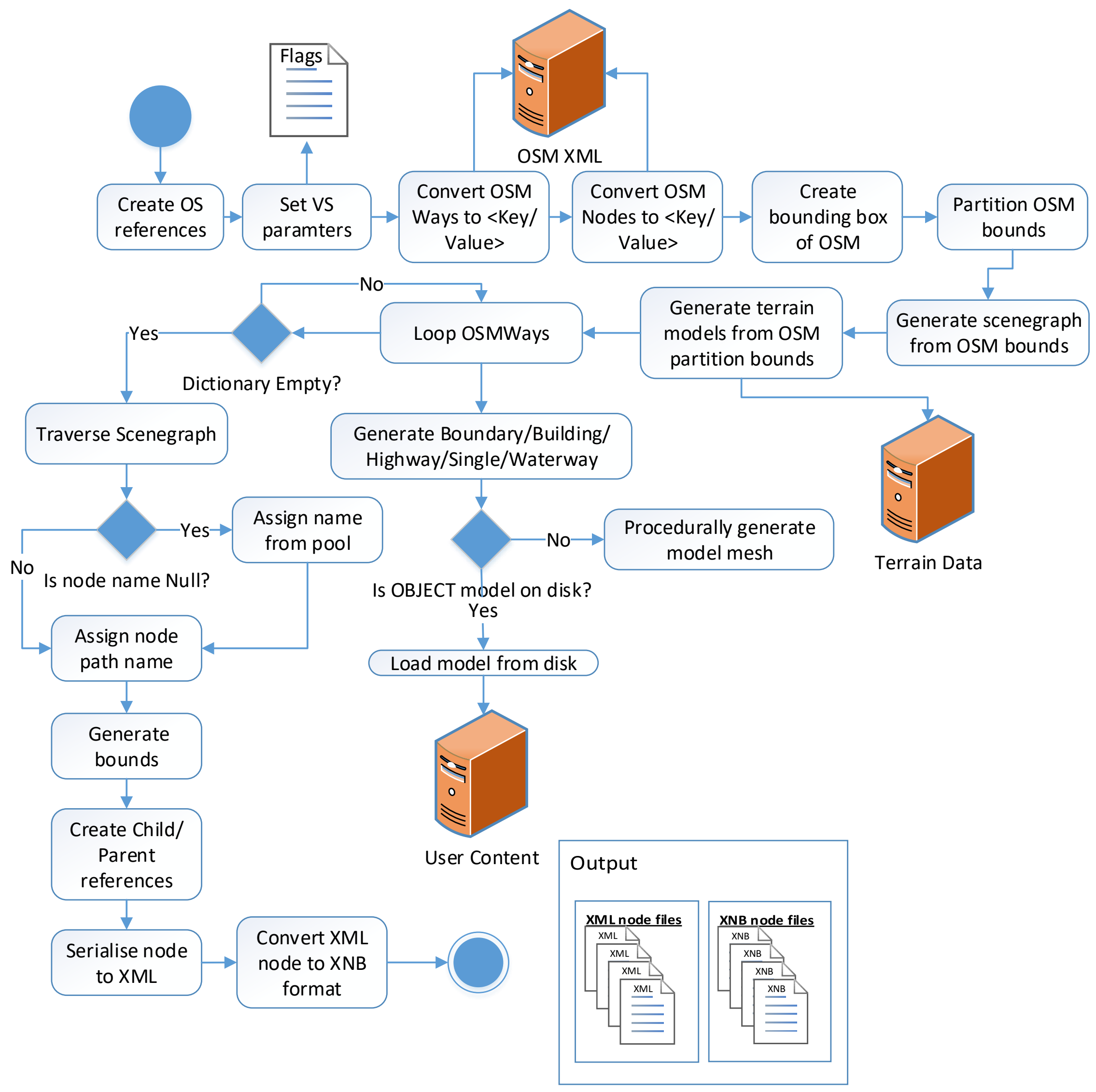


Figure 84 Activity diagram of algorithm for high-level overview of the pre-processor.

7.9.2 PVS Importer

To trigger the processing of the user selected OSM files, a custom file format is imported with a .pvs extension. Within the file, the OSM file is referenced, as well as data used within the algorithms. The additional data can be entered in as Visual Studio parameters, but if Visual Studio is not utilised, the data within the file is imported. This data consists of content root path, and other path directories. The Visual Studio parameters are explained in section 7.9.5. The imported object serialises from XML to a runtime object.

Originally we created many lines of text which encodes the flow of the processor, and tells the processor which LiDAR and OS terrain maps to create. We abandoned this approach to a procedural method which analyses the terrain maps which data from OSM coincides with; only then are terrain maps processed. This is due to our decision to convert and combine the LiDAR and OS maps, with data from OSM in a different program application.

7.9.3 Create OS Map References

The creation of the OS reference tile names at 500km, 100km, 10km, and 1km and additional 500m resolution is needed for quick look up and PG techniques for bounding box creations. Using the references, a scenegraph is generated as shown in Figure 56. Generating the scenegraph is procedurally done by querying the current tiles name. For example, creating a scenegraph from the name SJ3638, the name is processed through an algorithm which analyses the length of the name. If length is equal to 1 then tile is 500km. If length is equal to 2 then tile is 100km. If length is equal to 4 then tile is 10km. If length is equal to 6 then tile is 1km.

A switch-case is used with the length value of the name; branching to the correct bounding box PG procedure. PG is needed to increase processing speeds by removing the need of processing all bounds within the UK. It is also used to prevent memory wastage. Translation offsets are used to place the bounding boxes to the correct translation.

This process creates the full UK OS schema tile names at 500km, 100km, 10km, and 1km and additional 500m resolution stored as 1D List<string> structures.

7.9.4 Import 3D Model Mesh List

This is the importing of a text file which contains comma separated values of the OSMWay ID and the model name on disk. When imported the text file is converted to a key/value pair and stored in a C# Dictionary data structure.

Each OSMWay is checked against the list imported during processing and if the OSMWay is matched with the ID with the list, an external model mesh objects is imported, processed, and referenced in place of the procedurally generated process. This is explained further in a later section.

This improves scene realism due to current limitations of procedural techniques within the framework. An example to the document is shown in Table 14.

OSMWay ID	Model Name	Translation	Rotation	Scale
5099086	St_Georges_Hall	Vector3	Vector3	Vector3
25694396	Liverpool_Museum	Vector3	Vector3	Vector3
31294107	Our_Lady_&_Saint_Nicholas	Vector3	Vector3	Vector3
24611033	Liver_Building	Vector3	Vector3	Vector3

Table 14 Building model list key/value pairs used for importing premade model mesh objects

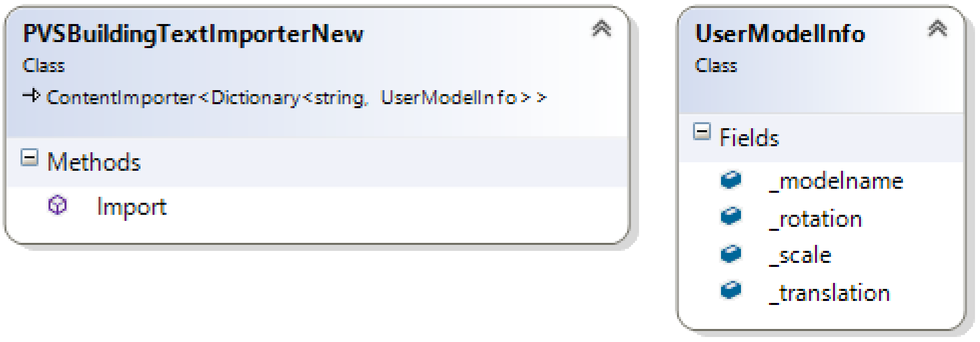


Figure 85 User Generated Model list class object and importer.

7.9.5 Set Visual Studio Input Parameters

Visual Studio allows the extension to import additional parameters into the processor. This allows users to modify the algorithms within the pre-processor. The parameters are explained in Table 15. They are primarily used within the Factory[56] classes which are used for the PCG of many similar types of model mesh objects; Building, Highway, Amenity, Boundaries, and primitives.

The parameters have default values and all processes are set to run. As stated within the specification section, the need for this technique is optional. It provides benefits to both development and user. Removing unneeded procedures will reduce processing time.

[56] https://msdn.microsoft.com/en-us/library/ee817667.aspx

Datatype	Name	Definition and Use
Boolean	Generate Boundaries	Flag to create boundary assets.
Boolean	Generate Buildings	Flag to create building assets.
Boolean	Generate Highways	Flag to create highway assets.
Boolean	Generate LiDAR Terrain	Flag to create LiDAR terrain assets.
Boolean	Generate OS Terrain	Flag to create OS terrain assets.
Boolean	Generate Single Nodes	Flag to create single nodes primitives
Boolean	Generate Waterways	Flag to create waterways.
Integer	Highway Interpolation Value	The interpolation value assigned to the highway generation process.
Boolean	Include 3D Model Assets	Flag to include user generated 3D models in place of procedurally generated assets.
Float	Texture Repetition Amount	How much a texture should rap a model mesh.
Float	Vertex Merge Distance	The distance between two vertex points. If distance between two points, the points will be combined.

Table 15 Visual Studio Parameter Input definitions and uses.

7.9.6 Import OSM XML to Class Object

The importing of OSM XML is imported using its own importer which returns a de-serialised XML object to a C# class object. Using this, the OSM objects can be queried and used within the novel algorithms of the framework. This process utilised the PVSXMLSerialiser Library and the OSM class objects.

During the serialisation, the .NetCoords library explained in section 7.3.4 is used to convert the Latitude and Longitude of the OSMNode, and the OSMWay to X, Y, Z coordinates. The centroid translation for the OSMWay is also calculated at this stage.

The OSMNode, and OSMWay are added to Dictionary<OSMNodeID, OSMNode> Dictionary<OSMWayID, OSMWay> respectively. This improves lookup speed from O(n) of the array, to O(1) of the Dictionary<Key, Value>.

The class diagram is shown in Figure 86.

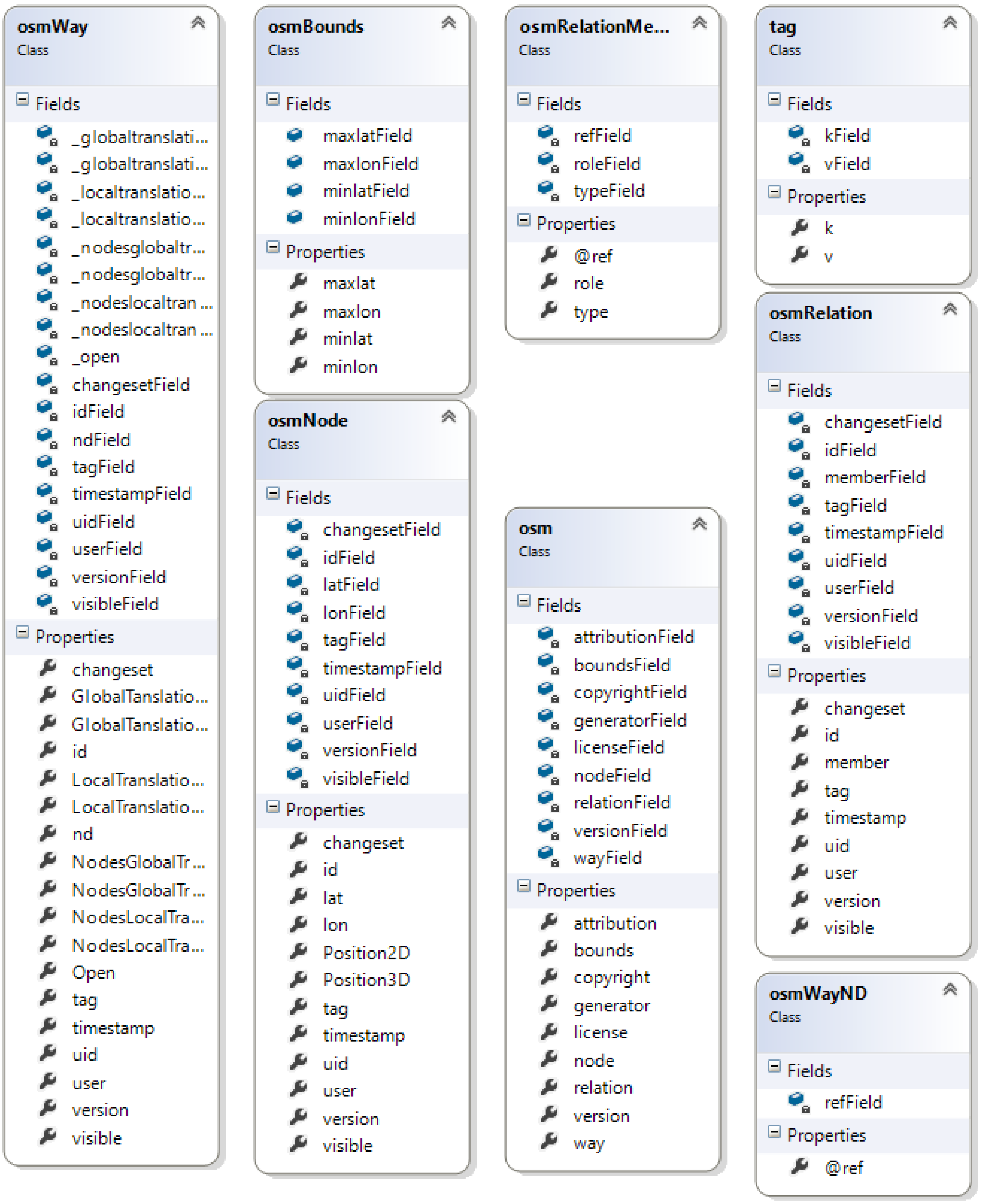


Figure 86 OSM Class diagram for OSM XML serialisation.

7.9.7 Create Bounding Box from OSM Bounds

A bounding box structure is created from the OSMBounds. The bounding box has basic collision techniques applied to them to determine the 1km^2 tile areas that it intersects with. This is explained in the next process section.

Due to the true bounds being different from the user selected area, reprocessing the bounds is needed. This is because user selected areas may select half a building, or part of a highway. OSM includes all of the data for any object intersected. Thusly creating bounds within house all OSMNode locations is needed. This will increase the size of the bounds, and potentially the size of the scenegraph and possible add additional terrain maps to be processed. The scenegraph within the pre-processor is created by analysing the OSNodes which the bounding box intersects at each level of the OS reference scheme. A Dictionary<tile name, bounding box> which contains all 1km^2 tiles names for which the OSMBounds bounding box intersects is generated. Using the Dictionary, the base of the tree is generated, as well as attached the empty OSM and terrain categorisation subtree to the 1km^2 OSNode. Figure 56 and Figure 57 show the tree base, and the OSM top.

With the names within a Dictionary structure, the keys are used to create terrain model mesh objects. If data is not available, a rectangular procedural polygon is created as a placeholder. The terrain mesh is textured using the corresponding OS textures.

Figure 87 shows the class diagram for creating the OS references needed.

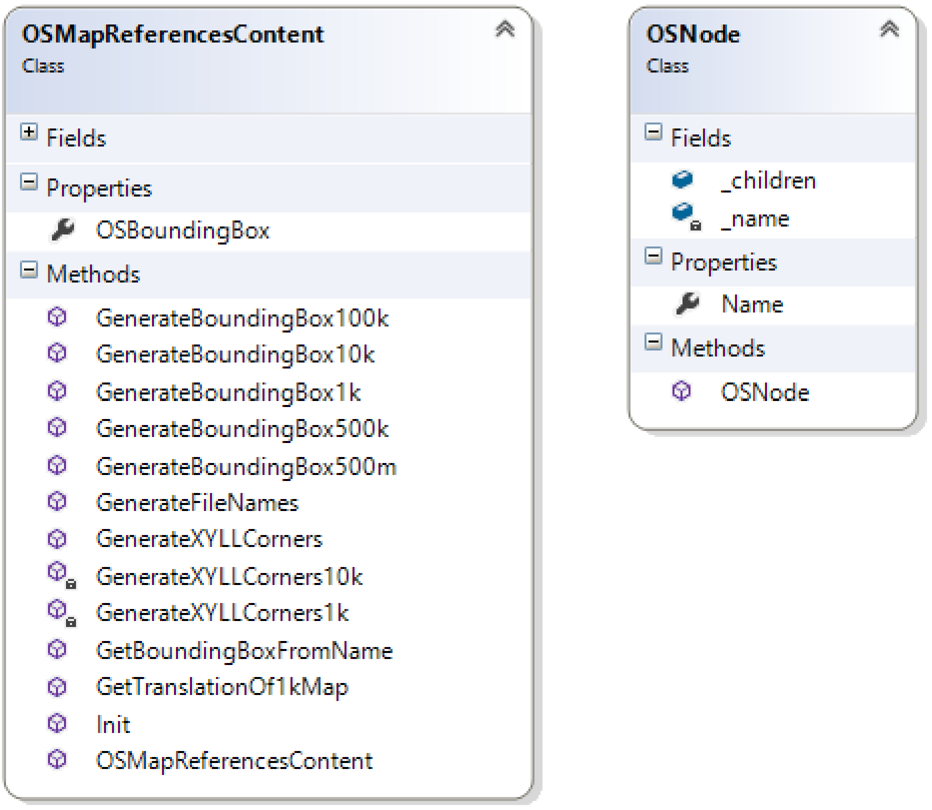


Figure 87 OSMapReference and OSNode class diagram.

7.9.8 Loop over Ways in OSMWay Dictionary

This process is where the individual model mesh assets are created by either loading user generated models, or PG techniques. The extraction and inference of OSM data is processes by the OSMInformationExtractor class, and the OSMHelper class shown in Figure 88.

OSMWay objects are iterated and processes in a specific order. The generation of boundary model mesh are completed first, then multi-part buildings, then single part buildings, highways, waterways and amenities. The boundary model mesh objects are created first as to allow the categorisation of unclassified assets such as buildings, or a highway. If a building has no classification, and intersects a boundary which is classified, the building object will inherit the boundary classification. This improves data representation and final visualisations as appropriate textures can be chosen.

Each OSMWay processed is checked against the user generated models assets on disk. If user generated content is available, then it is imported, processes, and PG can be skipped.

The use of the OSMInformationExtractor class and OSMHelper class are used to classify attribute data which is used to describe the assets being generated or loaded. For example, OSM allows any string representation to describe the colour of an object, or OSMNode. This ranges from the words "black", or the hexadecimal variant. The OSMInformationExtractor checks the input data and determines the validity, and returns a parse value, or a default value. This data is used to generate a PSVTag object. Each asset has its own PVSTag; PVSTagBoundary, PVSTagBuilding, PVSTagHighway, PVSTagAmentity. Figure 88 shows the currently implemented tags and the data parameters which they contain. They all derived from the interface class of PVSTagInterface. This allows updating and iteration of all types of OSMTag attached to any node within the pre-processor or runtime system.

The PVSTagHighway has derived children types which are used for the varying types of road categorisations.

The details of how each asset is created are presented next. The use of the Factory[57] design pattern is employed for the generation of model mesh objects. After an asset is generated, the OSMWay is removed from the Dictionary data structure.

[57] https://msdn.microsoft.com/en-us/library/ee817667.aspx?f=255&MSPPError=-2147217396

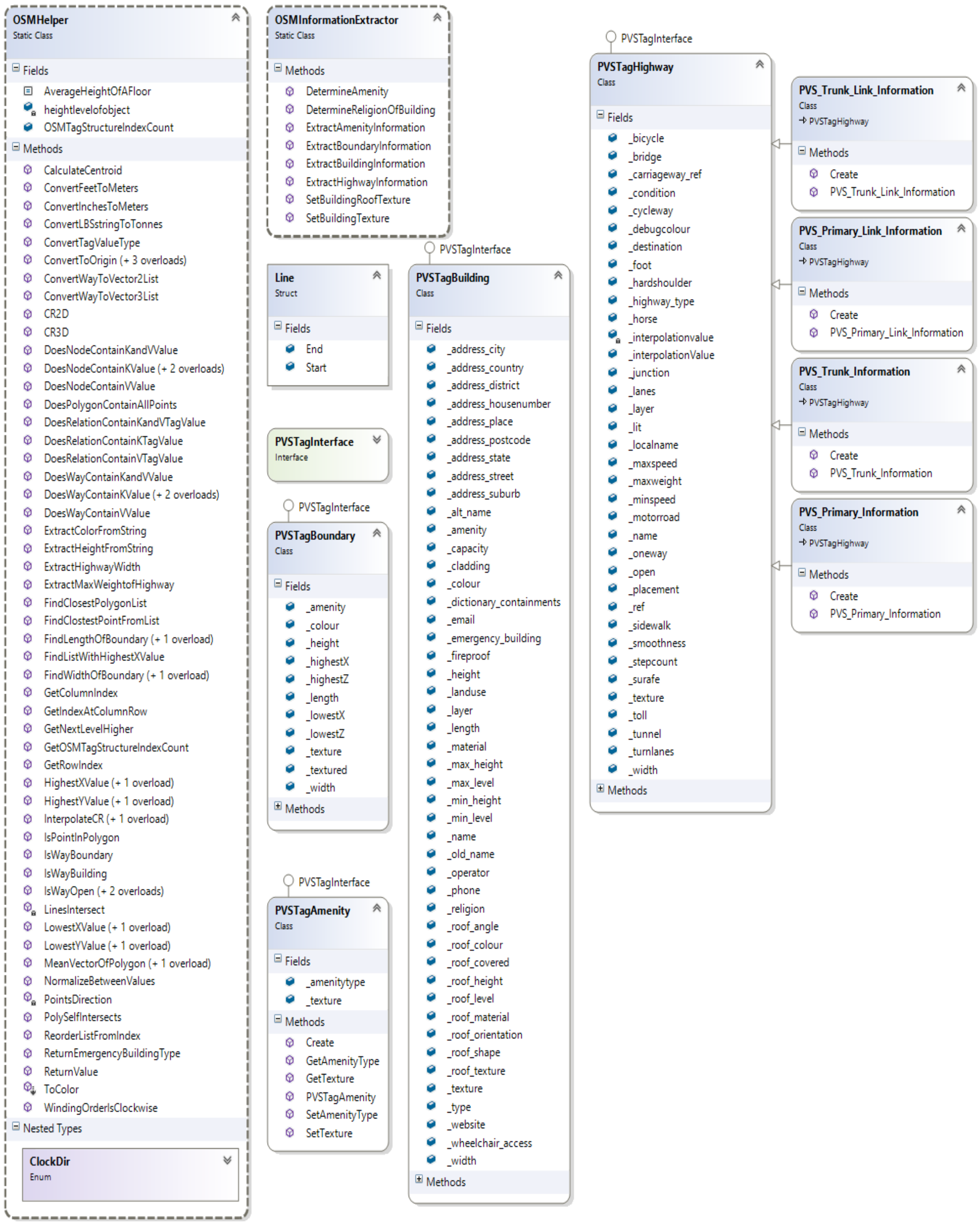


Figure 88 OSMTag class object which are created, and inferred from OSM data and attached to assets.

7.9.8.1 Generate Boundary Models

To determine if an OSMWay is of type boundary, the OSMTags attached to the OSMWay are checked for the key value of 'Boundary'. If true, the generation of the model mesh, and PVSTagBoundary are started. The check is only done on the Key, and not the Key/Value pair as is done for other assets. This due to the building types have values of ‘Boundary’, which will trigger the generation of a building mesh, and not a flat polygon.

The model mesh is a 2D polygon. The height translation will be set in a later process, placing the 2D polygon correctly on the terrain. The utilisation of the Triangulator library discussed in section 7.3.2 is used for converting the array of OSMNodes references within the OSMWay to an array of 2D locations in 3D space. These are the vertices. The Triangulator library produces the indices as well.

The classification of the boundary is checked. If it is the boundary of a hospital, police, fire station, education type, appropriate textures are applied to the procedural model mesh. If user generated content is available, the diffuse texture is not applied. The PG of UV coordinates are also created.

7.9.8.2 Generate Building Models

The iteration of OSMWays is undertaken to create the building model assets.

The building generation is built of two parts; checking if a building has multiple parts, and then checking if it is a single building. As stated, a relation is a linkage between different data types within the OSM database. Within the case of a building, an OSMWay represents the outline of a building and may have a relation to another OSMWay, or a number of additional OSMWays. These additional OSMWays which the first OSMWay are linked too, are connections between building parts. The most common is the relation between an Outer OSMWay type, and an Inner OSMWay type. This is pictured within Figure 89. The ordering is of the relations determines the structure of the building. In Figure 89, it creates a scenegraph structure which we exploit to create our building model mesh objects which have multiple buildings within one another.

OSM does its best to make sure buildings which contain multiple OSMWays comply by not having OSMWays overlap each other, or have the first OSMWay within a relationship be of type ‘Inner’. Errors are still within the database. Additional checks for these types of errors are needed. If an error occurs, we do not create any asset for that building. Future work will include messaging the user to determine what should happing in this case.

The information of that building is extracted and inferred to create a PVSTagBuilding object. Using this, the 3D building mesh can be created. The algorithm traces around the points of the OSMWay, creating plains between the current node and the next node vertically to the specified height of that building part. During this time, the UV coordinates are generated. Future work is to extract the building height data in an automatic process by utilising the LiDAR DSM as explained within our paper, “Mesh Extraction from a Regular Grid structure using Adjacency Matrix” in section 1.4.1, item

2. If no height value is available, and cannot be inferred, the default height will be the height of a single floor which is 12 feet or 3.6576 meters as stated within the design section.

The roof structure is created using the same techniques utilised within the boundary generation process. The Triangulation library allows for holes to be cut out of the polygon. This allows rooftops to be generated for both a multi-part building, and a regular building.

The addition of a randomly added height value between 0 and 1 is added. This adds variation to the rooftops of the buildings which improves realism within scenes by modifying the output of the lighting calculations.

The algorithm stated allows the PG of any building within the OSM database, and thusly any building structure in the World. Initial prototypes have successfully created scenes from Manhattan New York, and Liverpool, Manchester, and London in the UK. Figure 90 top left is the OSM map of Manchester UK and the procedural scene produced top right, bottom left is Liverpool UK, and bottom right is Manhattan New York USA.

To categories the model mesh objects into specific nodes which is stated by the specification, the PVSTagBuilding object is queried to state whether the building is a hospital, police, fire station, military, educational, and attached to the appropriate PVSNode within the scenegraph. If not any of these they are attached to the ‘other’ PVSNode which is used for the classification of all other types of buildings.

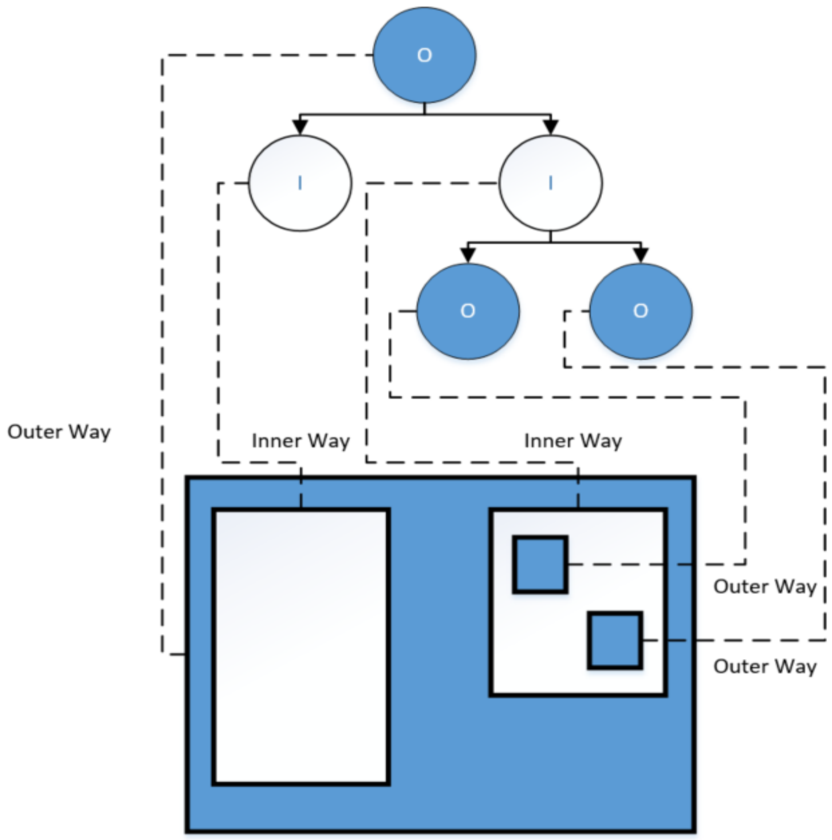


Figure 89 Building Relation explanation. Shows Outer OSMWay, Inner OSMWay, and Outer OSMWay. The order determines which buildings are within one another.

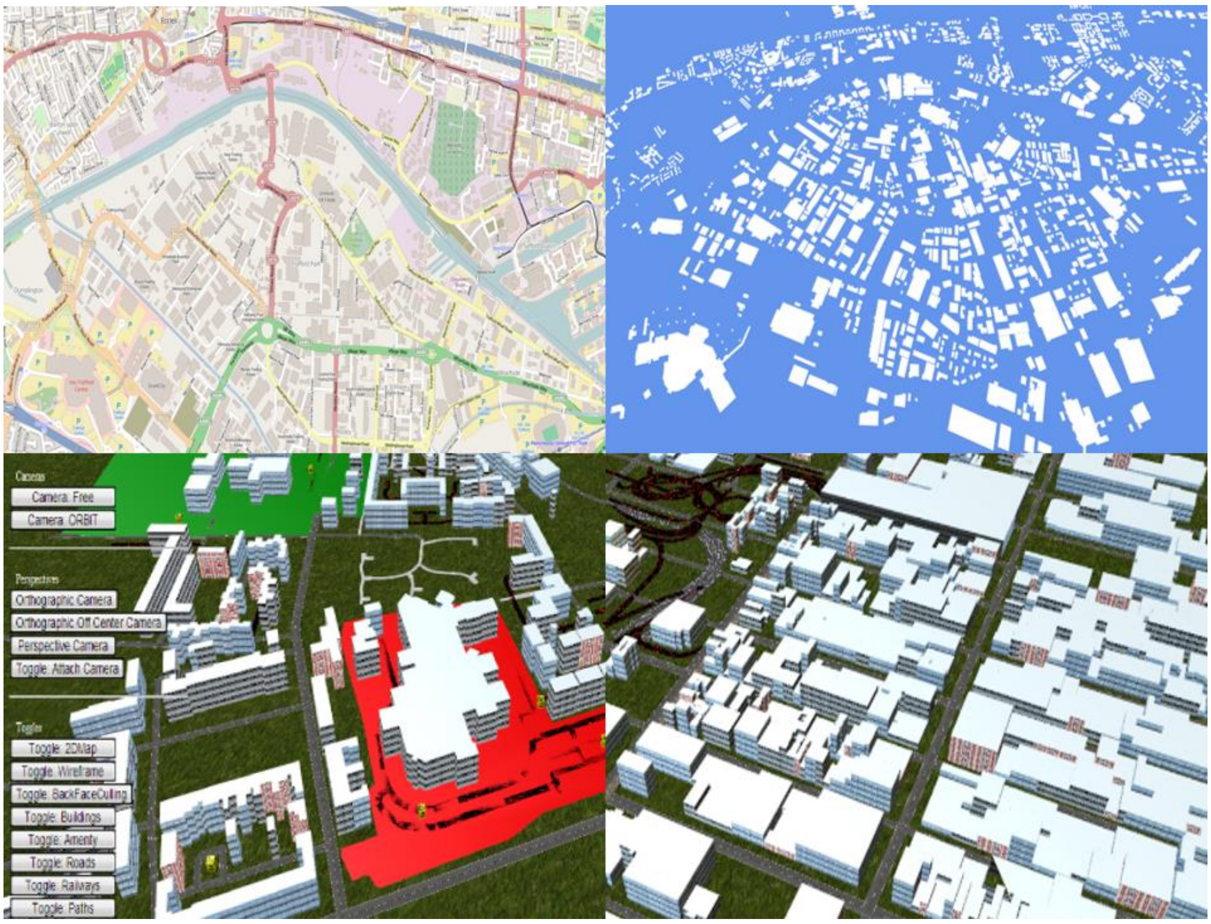


Figure 90 Prototype building images. Top right is Manchester UK with accompanying OSM screen capture to the left. Bottom left is Liverpool UK specifically Liverpool Women's Hospital highlighted with red boundary. Bottom right is Manhattan New York. Top left image obtained from https://www.openstreetmap.org/search?query=trafford%20centre%20manchester%20uk#map=14/53.4719/-2.3126

7.9.8.3 Generate Highway Models

If an OSMWay is of type 'highway', then a highway model mesh structure is created as well as the PVSTagHighway object. Due to the large amounts of highway structures (Roads, Link Roads, Paths, Waterways, Specials etc.), the PVSTagHighway has alternative tag structures which have default values applied to the data, dependent on the classification of the highway. Many highway structures are unclassified. If they are classified then much of the information needed is missing. This information may be the width of the highway, number of lanes, road markings, names etc. Each PVSTagHighway object is created with default values. The information extracted from the OSM data-base overwrites this default data. The default data is populated by research, and recommendations from professionals, and OSM.

The use of Catmull-Rom interpolation in conjunction with the 'HighwayInterpolationValue' are used to generate additional points along the OSMWay to increase the model tessellation. This improves realism by removing sharp corners which come with working with low resolution highway representations. The higher the ‘HighwayInterpolationValue’, the increased number of vertices will be created and the smoother highway model mesh objects will be.

The interpolation of points are used as the centre of the highway structure along the path. Using the mathematical function of the Cross-Product[58], the edges of the highway structure are created by taking the width of the highway from the PVSTagHighway object, halving this value, and multiplying the Cross-Product vector direction by this amount. Negating this process creates a point the opposite side. Vertices are created using these two Cross-Product generated points. Polygons are created like this through the highway structure. See Figure 49 for visual representation of this algorithm.

The model mesh object is attached to the appropriate node within the scenegraph by classification from the PVSTagHighway.

7.9.8.4 Generate Single Models

We currently use procedural cube mesh objects which have varying textures applied to them for depicting certain areas, or assets; bars, banks, parking etc.

The single nodes have attached the *PVSTagAmenity*.

7.9.8.5 Generate Waterway Models

The water OSMWay models are a cross between a boundary and a highway. The boundaries can be lakes of water, whereas rivers are classified as highways. We have created methods to determine the classification of the waterways or water boundaries and place them within the scenegraph accordingly. They have the same processes of generation as that of a boundary model mesh or a highway model mesh dependent on their type. The rendering technique and textures are predefined for this type of object.

7.9.9 Create and Assign PVSNode attributes for serialisation

A PVSNode is created with a unique ID. If a PVSNode does not have a name attached to it, the ID will become its name. The name otherwise will be populated by the OSM data. The importance of the name is for reference by other PVSNodes; its parent, or its children. This is needed, along with the full path of the PVSNode within the scenegraph due to the inability for the PVSXMLSerialise to serialise cyclic references. The path name is built by traversing the scenegraph to the root node, returning a concatenated string of PVSNode names. It is this path which the serialised XML file, and

[58] http://mathworld.wolfram.com/CrossProduct.html

the formatted XNB file is save as. An example of this is;
“world.s.sj.sj38.sj3189.osm.buildings.other.Royal Liver Building”.

The bounding boxes are generated by combining all bounds higher in the scenegraph, starting with the model part bounds, and reclusively back tracking to the base of the scenegraph combining the bounds together.

Once all attribute parameters are referenced between parent and child relationships of the scenegraph, the XML serialisation of each PVSNode of the scenegraph is processed. As stated, the MS Build process XNA uses allows only a single input, single output pipeline. XML serialisation side steps this drawback. The files are reprocessed to write the nodes to the optimised XNB format.

7.10 Conclusion

To conclude this chapter, we have presented the implementation of internal APIs’ used within the PVS framework. We have stated the software of choice and presented the advancements which MS Build provides PVS to compile objects pre runtime. External animation and interpolation, triangulation, projection conversion of Dot.NetCoords, and the Microsoft LINQ language APIs’ are also presented for their use within the pre-processor as well as the runtime application. Procedural generation algorithms are presented which in turn check error prone big geospatial data, for the generation of 3D model mesh data. The pre-processing of this data allows for improved loading speeds at runtime. The internal APIs’ of PVS Camera Library, PVS Input Manager, PVS XML Serialiser and OSM Enumeration are also used within the runtime application processing.

The next chapter discusses the implementation of the runtime application, specifically the algorithms used for processing and rendering, combined with the IVI.

8 Processing and Rendering with Interactive Visualisation Interface

Within this section of the chapter we discuss the implementation of the runtime algorithms used visualise the processed big-geospatial data in a real-time application. Interaction of the application is performed through the IVI which allows dispersal of shader index values to change a scenes visual parameters within milliseconds. The IVI also allows data searching on a scene basis, searching for user defined queries and applying user generated spatial or visual controllers to depict objects from a breath of big geospatial data renderings.

8.1 PVS Runtime

The runtime system is built of many components, class objects, managers, hierarchies, helper classes and functions. Figure 32 shows the high-level design and interconnection between objects within the runtime system. The system resolves around PVSSpatial and its inheriting object of PVSNode which creates the scenegraph structure which all assets will be attached, or loosely affiliated with. An example of an affiliated object is the Light Manager which is explained next. We also explain the implementation of the components of spatial controllers, material controllers, model and model part objects, with the accompanying material object each model part contains. The material is used to pass parameters to the IASF which we will discuss with the accompanying GPU render state selection algorithm. The IVI system is also explained for its role in varying input and scene visualisations.

8.1.1 PVSSceneLightManager

The PVSSceneLightManager is responsible for the creation, updating, and rendering of 3 light structures. A light structure holds the relevant information needed for rendering; position, direction, colour, and linear, constant, and quadratic attenuations. These are the global lights which can be stored within each of the model parts PVSMaterial object as a reference, or the model can choose to use its own light structures, or copy light structures of the scene. This provides flexibility with the lighting parameters of a scene. Instead of creating a material controller which changes the material properties of selected models, a PVSLightDescription can be created and attached to the PVSMaterial as a reference. Changing the parameters of the PVSLightDescription will change the renderings of all models which reference that PVSLightDescription. This allows scalability and flexibility to scene manipulation and visualisation.

The PVSSceneLightManager has a reference to the base PVSNode of the scenegraph. This allows the PVSSceneLightManager to manipulate the scene as needed. An example of this is forcing the current 3 PVSLightDescriptions onto every model within a scene; unifying the visualisation of the scene.

The PVSLightHelper is a static class which contains regenerated PVSLightDescriptions.

The PVSLightDescription contains attributes needed to for lighting calculations, and to manipulate the light parameters by input controls. Figure 91 shows the PVSLightDescription structure. It contains attributes which are used to describe both directional and spot lights. As stated, a point light is not utilised due to the point light equation can be encapsulated into the spot light equation. The attributes are duplicated on the IASF effect HLSL file except the enabled Boolean and light type enumeration. These are utilised CPU side to select the shader index of the IASF array lookup variable. This is explained more in section 8.5.

Due to the scene allowing only 3 lights, the encoding of the lights within the IASF needs to be considered. Please see section 8.5.2.

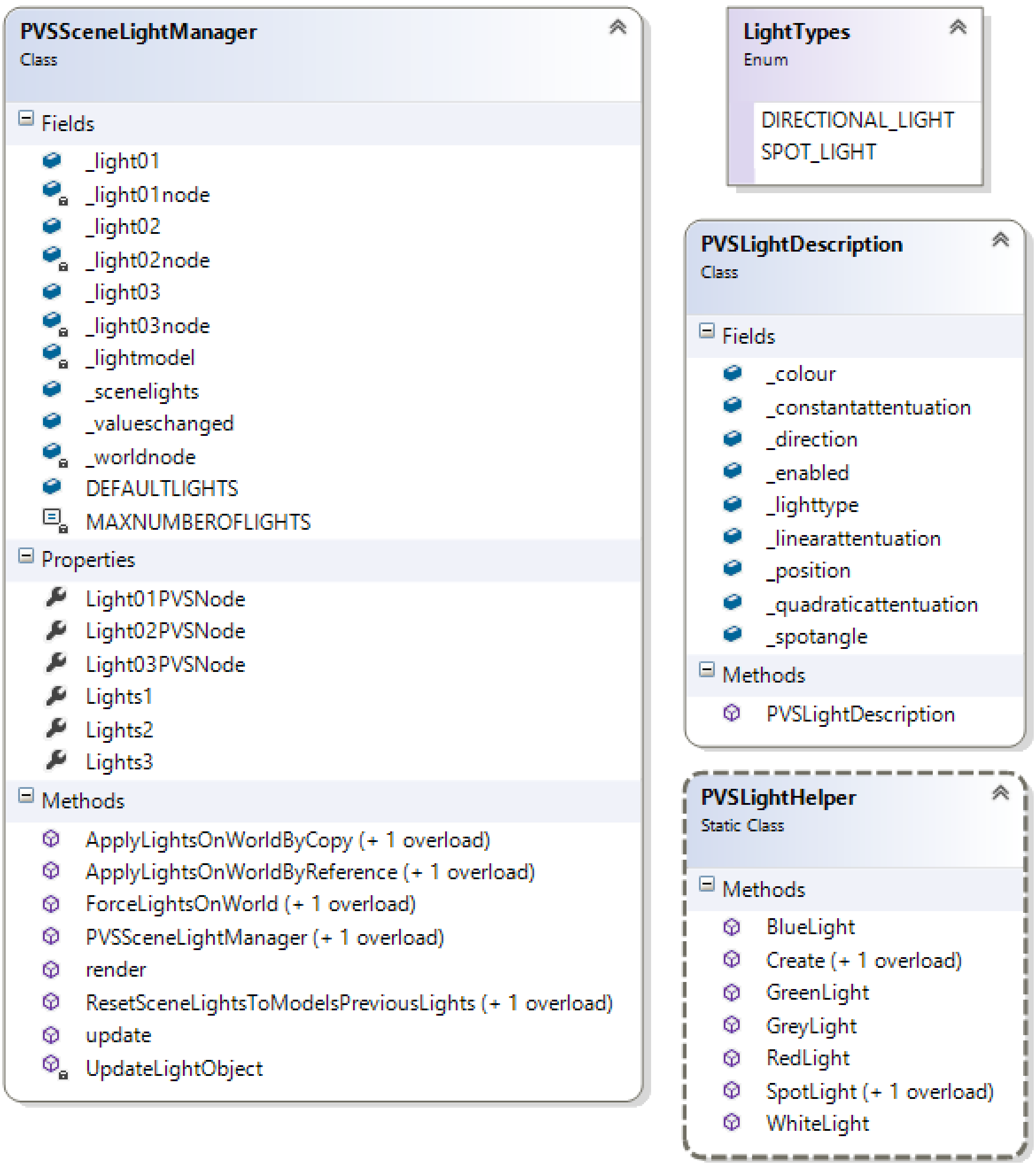


Figure 91 PVSSceneLightManager class diagram and supporting class objects.

8.1.1.1 Light Data-Structure

As with the material structure, we have created and used a light structure. This structure contains parameters pertinent to describing common light definitions. These lights we cover are; ambient light, spot light, and point light. We state that a spot light and a point light calculation can be collapsed into one another. The data structure contains; position, direction, colour, spot angle, a constant attenuation, linear attenuation, and quadratic attenuation. These are stored as an array within the effect object. During updating and passing of light data to the effect object from the CPU to the GPU, the lights are reordered dependent on which lights are enabled or disabled.

8.1.2 Middle-out Loading of the Scenegraph at Runtime.

In the runtime simulation, the loading of the scenegraph is created using a middle-out technique. A user can select a 1km^2 tile with the node name of “world.s.sj.sj38.sj3189”. Its parent node of “sj38” with path name of “world.s.sj.sj38” is loaded next, and recursively loops until the node being loaded does not have a parent. The node which does not have a parent will be the base of the scenegraph. The node of “sj3189” will have an array of child node path references which it will load accordingly. Because the parent node and children nodes of the node being loaded are stored in separate references, the scenegraph structure can be generated using a multithreaded approach.

With nodes being loaded from the middle to the base of the tree, the children of the nodes to the left of the currently selected node will not be loaded. This is to reduce the number of objects loaded. Figure 92 shows the middle out loading starting at the SJ3189 node. The green nodes are the nodes which represent OS nodes; 1km^2, 10km^2, 100km^2, 500km^2, and the base node which represents the world. The nodes in red are ones which may be generated and stored but are not loaded. The blue nodes represent the OSM objects.

If additional nodes are loaded, they are checked to see whether they are held within the scenegraph. This allows for adjacent tiles to be loaded, and removed with minimal of processing.

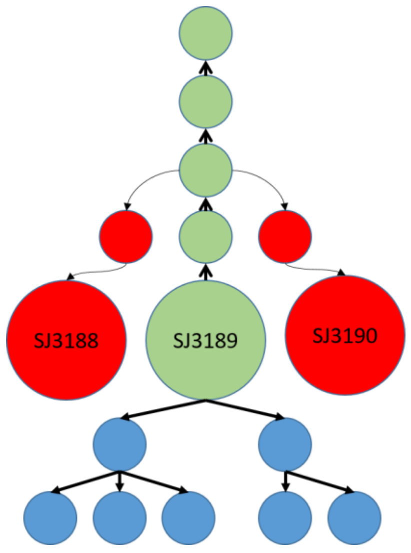


Figure 92 Middle out loading. Green represents the OS nodes of the scenegraph, red are nodes which are not loaded, and blue are nodes which represent the OSMNodes.

8.1.3 PVSSpatial and PVSNode

The PVSSpatial is a node object which the scenegraph is built from. It inherits from 3 interface classes of;

- IInputUpdatable – inherited classed implement an update function which takes in the time-per-frame, the total passed time, the PVSInputManager, the current cameras view matrix, and the current cameras projection matrix. This guarantees the classes which utilise this interface can update the most common user controlled objects.
- I3DRenderable – demands inherited classes implement the render function which takes as parameters the current cameras view matrix, and projection matrix. This will be used to render objects without passing a reference of the currently used camera. Technically the camera may not even be used and a custom view or projection matrix may be entered.
- I3DCamRenderable – demands inherited classes implement a rendering function which takes as a parameter of the IPVSCamera interface. This allows the passing of many of the implemented camera object types. The view and projection matrices will be extracted in the calling function. The reason for multiple render functions is because not all cameras utilise the IPVSCamera interface.

Figure 93 shows the class diagram of PVSSpatial and PVSNode.

The PVSSpatial object stores a reference to the current graphics device which is used for rendering. The reference pointer improves efficiency within the render function than passing the object through as a parameter. The PVSModel may be stored as a null attribute. The original translation, rotation, and scale vectors are stored and these will not be modified. The modification to the translation, rotation, and scale will occur on the separate variables references. This removes the need to load the data from file when the scene is set back to an original state. The PVSSpatial object stores the current world matrix which is a combination of the translation matrix, rotation matrix, and scale matrix used for rendering the object within the world. This data is normally processed on the GPU, but offsetting it CPU side improves rendering framerates. The identity matrix is used for normal mapping shading techniques. The class stores a reference to another PVSSpatial object which will be its parent in the scenegraph. If a parent reference is null, the assumption that it is the base of the scenegraph can be made. The PVSSpatial also stores a List<IControllerSpatial> which stores the spatial controllers attached to the object. Within the update call, the iteration of the controllers will be updated which modify the parameters of the PVSSpatial object they are attached to. When a spatial controller is attached, the function attaches a reference of the PVSSpatial object to the controller. This allows communication through the objects. A Boolean value is stored to state if it uniformly scaled. If it is not, then further calculations are needed as non-uniform rendering of models using normal mapping techniques will present issues. The PVSSpatial has Boolean flags to skip the rendering, and updating calls. This is used to cull objects from a scene, but not to remove them from memory, nor from the scenegraph.

An initialisation Boolean flag is triggered after instantiation. This function call creates references from the object to the PVSMaterial object held within the model parts of the attached model. This allows communication between PVSMaterial objects and the hierarchy it is contained in. Some material controllers rely on communication between the PVSMaterial, and the PVSSpatial object, thusly this reference between them allows for skipping of additional function calls between objects within the scenegraph between PVSMaterial, to model part, to PVSModel, to PVSSpatial object.

An additional Boolean flag is used to state that the PVSSpatial has not been modified. This will skip the updating of parameters, and recalculation of the world matrix. This is also communicated to the PVSMaterial object attached to model parts which will not recalculate lighting parameters, or other material properties.

The PVSSpatial node is intended to act as a leaf node of a scenegraph. If a node is intended to contain children, then the class is inherited by the PVSNode.

Using the two classes of PVSSpatial, and PVSNode, large scenes can be created, updated, and visualised.

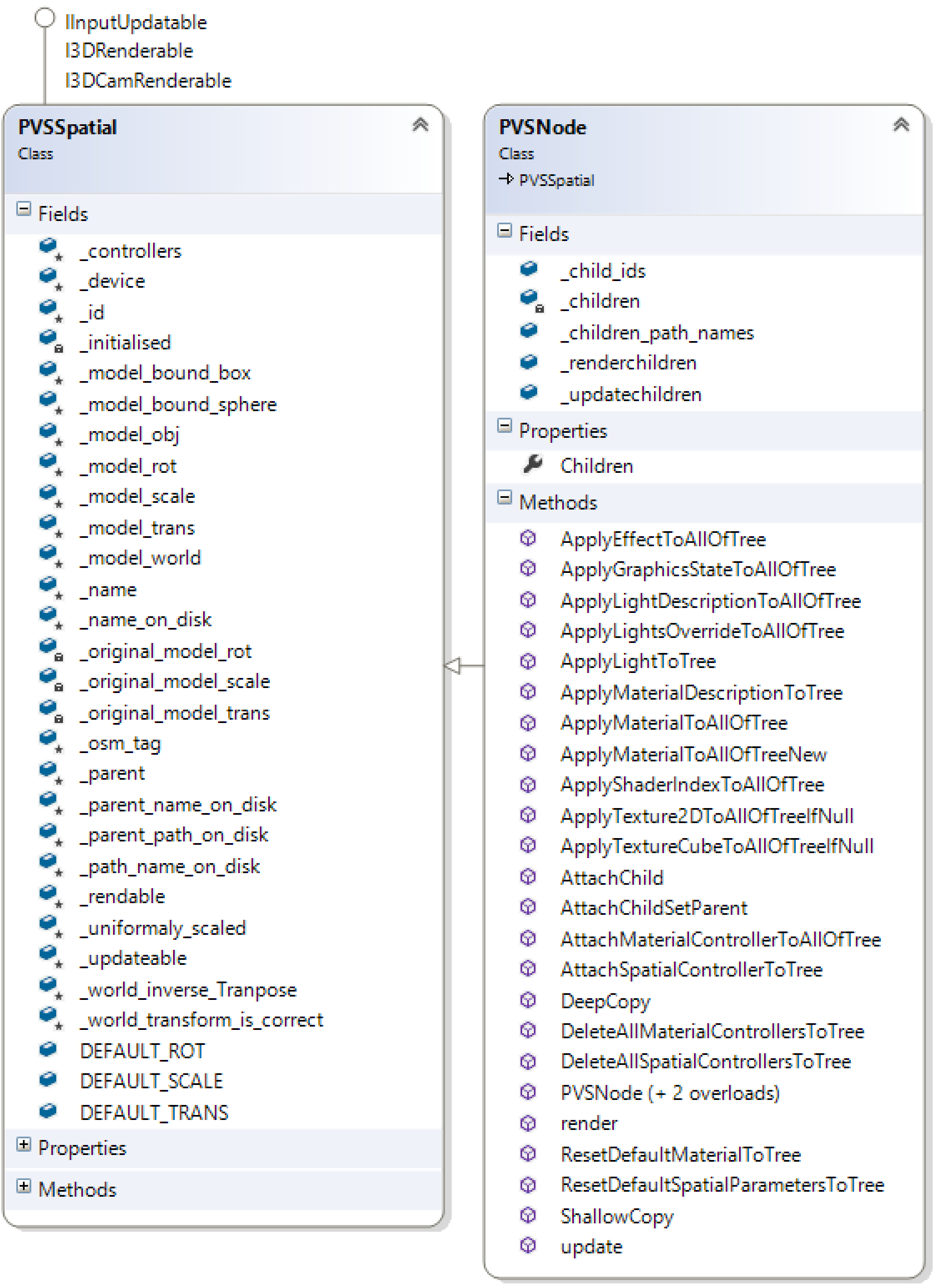


Figure 93 PVSSpatial and PVSNode class diagram.

The initialise method is used to initialise the PVSSpatial node by attaching the graphics card for rendering, attaching the node to the model mesh object, and attaching the model parts to their respective material objects. It internally updates its bounding box and sphere volumes, and combines its parents world matrix with its own as to translate, rotate, and scale the node properly, setting the '_world_transform_is_correct' to false to force an update of the complete tree on its first update. This creates a new reference of the controller list, setting whether the object is uniformly scaled, and then setting the 'initialised' parameter to true.

8.1.3.1 Additional Functionalities of the PVSSpatial and PVSNode Object

The other functions within the PVSSpatial class consist of;

- Attach controller – attaches a spatial controller to the list of controllers controlling this node. The parameters passed allow the user to check if a controller of the same type is contained. If it is already contained it can either be added again or not.
- Remove controller – if the list of controllers contains an instance of a controller then it will be removed. This does not work on the 'type' of a certain controller, only the referenced object.
- Check controller list contains controller type – this function checks if the controller list contains a certain 'type' of controller and return true or false.
- Apply material to all model parts – this function checks if a model object is attached; if true then it will iterate the model parts within the model and apply the material to each.
- Apply material controller to model – if a model object is attached this will iterate over the model parts and attach a material controller to each part.
- Reset default material – each model part contains the currently used material and its original material loaded from disk. This function will reset the currently used material to its original.
- Reset material controllers – this function clears the list of material controllers for each model part of the attached model object and removes the no default controllers.
- Delete all spatial controllers - removes all the spatial controller attached to the node. This does not reattach the defaulted needed controllers.
- Reset default spatial parameters – this resets the currently used parameters to their original values. This allows the user to translation a node and then to reposition it to its original parameters.
- Apply lights to model – this function applies user defined lights and applies the lights to each part of the model object.
- Attach world matrix – attach a new world matrix.
- Attach parent – attaches a parent reference to this node
- Attach parent set child – attaches a parent reference to this node, and then attaches the current nodes to its parents' child list.

- Restore material to original – restores each model parts material to its original parameters.
- Delete all material controller – deletes all material controls within each of the model parts within the attached model object.

The virtual functions which are intended to be overridden by the inheriting class of PVSNode are listed;

- Update – explained in next section.
- Render – explained in next section.
- Attach spatial controller to tree – attaches a spatial controller to every node within the scenegraph.
- Attach material controller to tree – attaches a material controller to all model parts of the model held in each node within the scenegraph.
- Apply material to all of tree - attaches a material to all model parts of the model held in each node within the scenegraph.
- Apply effect to all of tree – applies a HLSL effect object to each model part of the model of each node.
- Apply graphics state to all of tree – applies a graphics state enumeration value to each model of edge node.
- Apply shader index to all of tree - applies the rendering shader index to be used by all model parts of each node within the scenegraph.
- Reset default material to tree – resets all material object to its original parameters for the model parts of each model of each node within the scenegraph.
- Delete all material controllers to tree – deletes all material controllers within each model part of the model in each node of the scenegraph.
- Delete all spatial controllers to tree – deletes all spatial controllers within each node of the scenegraph.
- Reset default spatial parameters to tree – this deletes all the spatial controllers within each node of the scenegraph and then applied the default needed spatial controllers to each node of the tree.
- Apply lights to tree – attaches lights to each model part of the model attached to each node of the scenegraph.

There are two virtual functions which are intended to be overridden within the derived class of PVSNode. These are the update function, and the render function. On the update of a PVSNode, its spatial controllers are updated. If the PVSNode needs updating, the spatial matrices will be updated. The update will update the material parameters of all model parts attached. Each update call on a

PVSNode will call into its children's update functions. This is a recursive function. This allows the updating of the scenegraph. A check is made during the update to see if the PVSNode should update its children, or itself. This allows pruning of branches of the scenegraph. The same technique is used for the rendering of the model data within the PVSNode; pruning branches of the scenegraph which do not need to be rendered.

8.1.3.2 PVSNode connections and links through framework

Figure 94 shows the reference connections between the objects within the scene with the PVSSpatial and attached objects of PVSTag, PVSModel, model parts, and PVSMaterial as well as the spatial controllers and material controllers.

The culmination of these references allows for the rendering of large scenes, and customisable controllers to be attached to both PVSSpatial node objects, and the material object. This allows for every object be it spatial parameters to the visual representation of objects, to be modified, animated, or user defined.

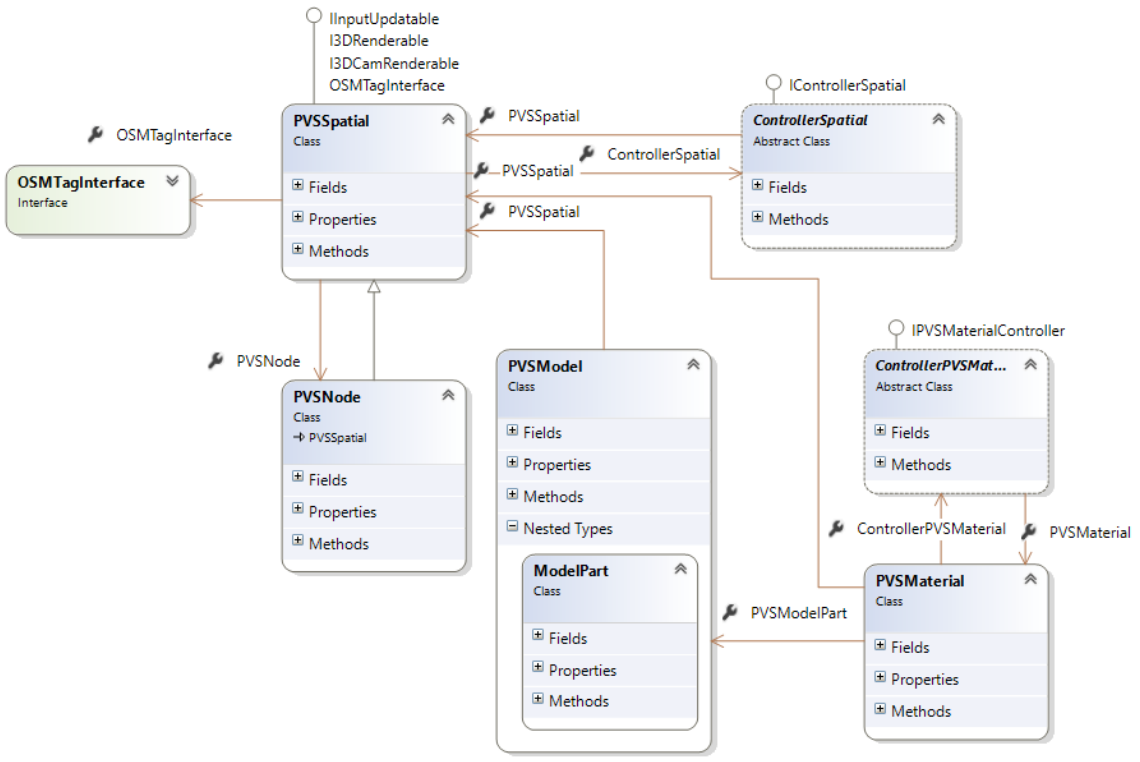


Figure 94 Referenced objects between each other.

8.1.4 Controller: Spatial and Material

We discuss the controllers of both spatial and material due to their similarities and main difference is the object for which they are attached; PVSSpatial or PVSMaterial.

Figure 95 and Figure 96 show the class diagrams for the material and spatial controllers. Controllers come in two forms; a tweener controller, and a non tweener controller. The use of the interface which controller inherit from is to allow storing of various types of controller within a single array structure, and allows updating of all controllers by calling the overridden update function.

The controllers have references to the objects for which they are attached. If additional references to objects are needed, the constructor is extended to input these references. As a minimum, the application timer, and input manager is referenced.

The tweener controller has a reference to a tweener object which is used for interpolation/animation. See section 7.3.1 for an explanation of the animations and interpolations provided by the tweener object. It also has an array of parameters passed through the constructor and a string name reference used to connect to a property of the attached object. The use of the C# *params*[59] keyword is used to pass through an array of objects. For example, an array of numerical values. The values are used in tandem with the property name. The property name will be used to extract the corresponding property from the attached class object type. For example, passing the string representation of DiffuseColour will query the attached object type, in this example the Material object, and return a Get/Set property, allowing the getting and setting of the diffuse colour variable which is a Vector4 type holding the RGBA values used on the GPU. The array of integers act as the indices to the RGBA or XYZW of the Vector4 object. A switch case is used to state which parameters of the property are to be modified. A switch statement is used to allow combinations of the parameter indexes to be used. i.e. if [0, 1, 2, 3] is passed through as an array, then the XYZW components of the Vector4 will be updated according to the interpolated value of the tweener. If [0, 3] are passed, then only the X, and W components are altered.

This functionality allows for any property of both the PVSSpatial, and the Material object to be modified, updated, animated, and interpolated. The PVSSpatial, and Material objects can have any number of controllers attached to create interesting effects, and encode user defined data onto single or multiple objects within a visualised scene. For example, objects which need to be brought to the attention of the user can be animated by pulsating the scaling of a building, as well as pulsating the R component of the diffuse colour.

[59] https://msdn.microsoft.com/en-GB/library/w5zay9db.aspx?f=255&MSPPError=-2147217396

We highlight the alternative use of the controllers next by stating the connections between the controllers and other components such as the 3D camera system for the reduction of processing and parameter passing by only updating and recalculating world matrices if either the camera or the PVSSpatial have been modified.

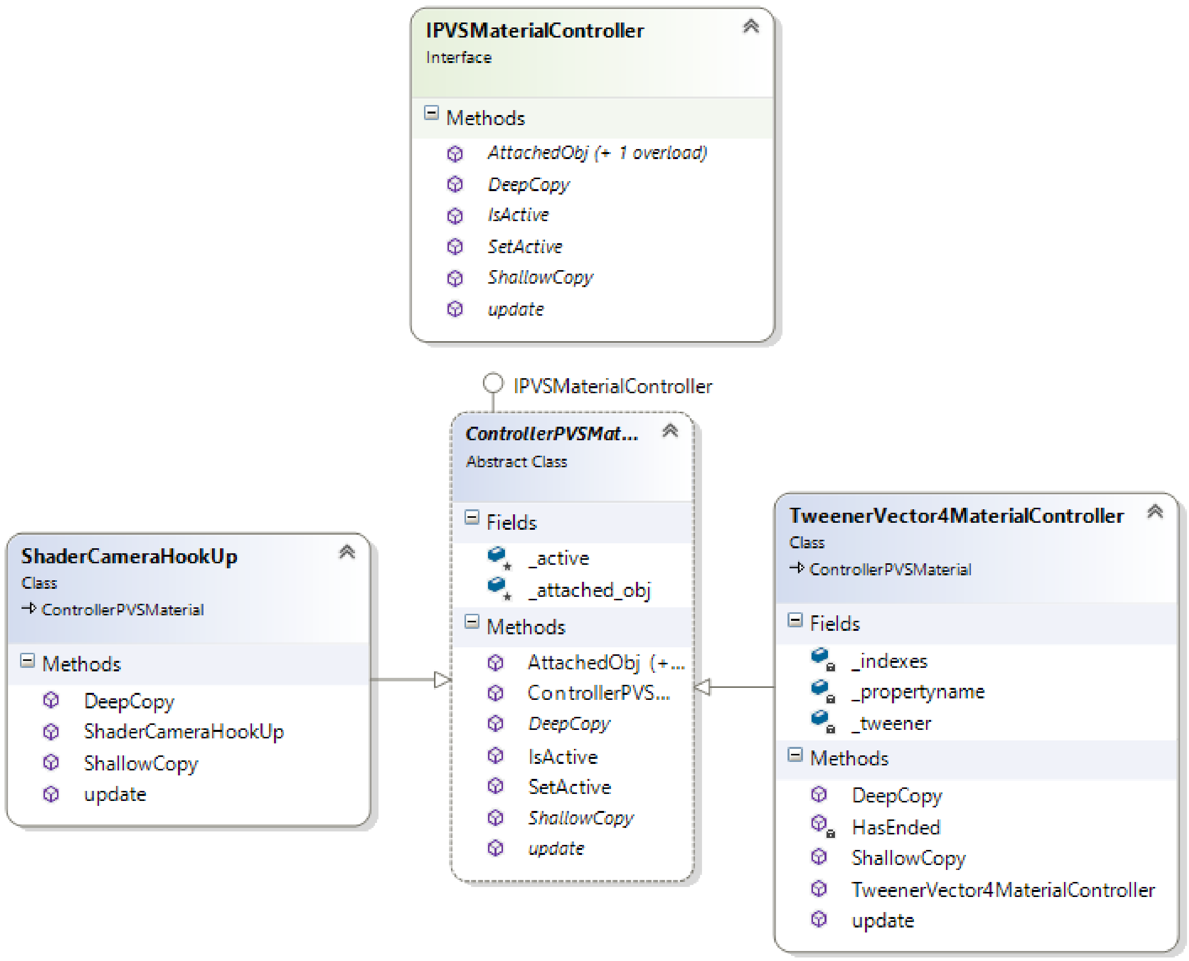


Figure 95 PVSMaterial Controller class hierarchy for the two default controllers for each model part of each model object.

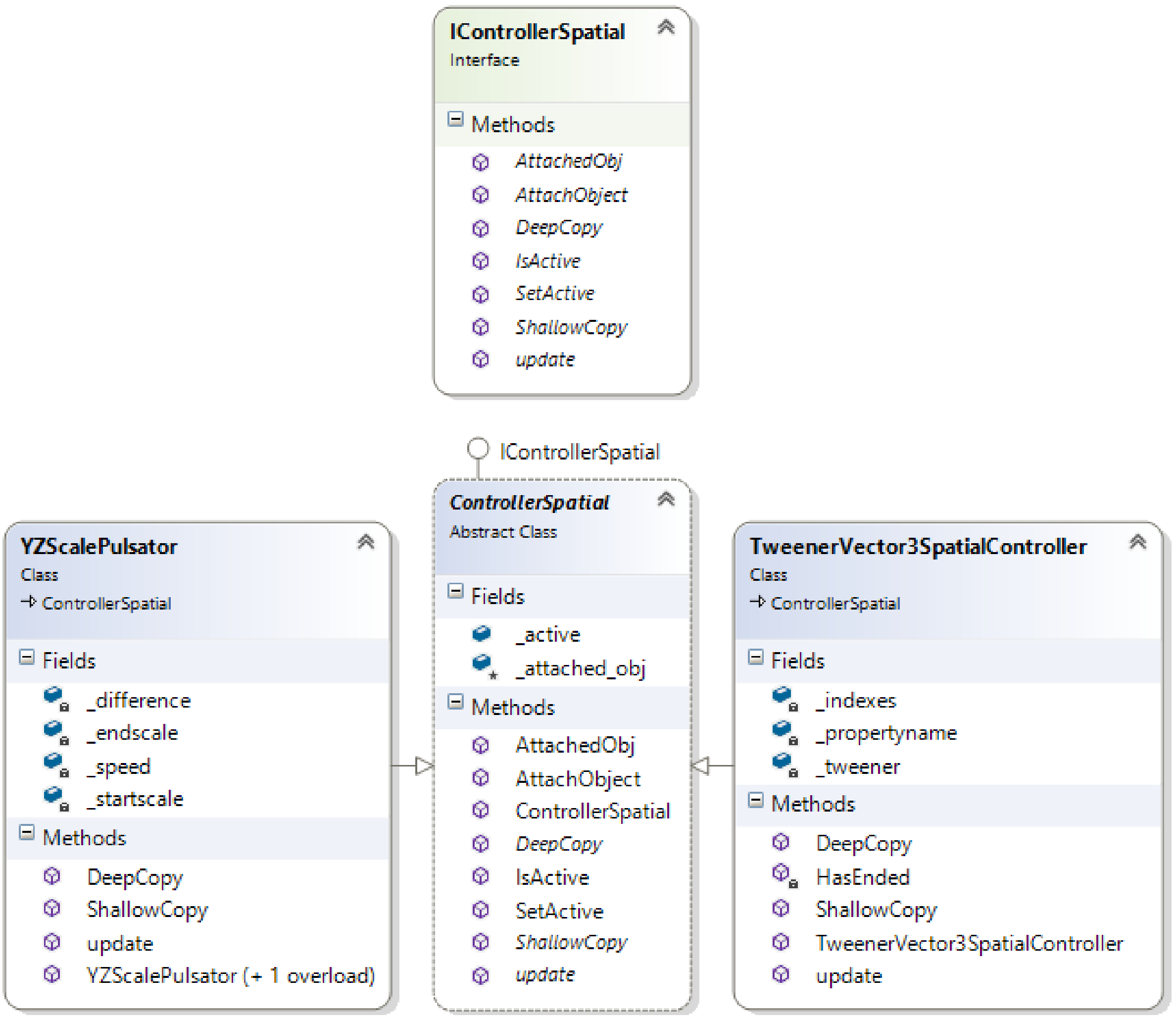


Figure 96 Spatial controller class diagrams.

8.1.4.1 Shader Camera Hook-up

This class is responsible for setting Boolean properties within the Material object. It checks the static Boolean '*IsCameraDirty*' contained and set within the PVSCameraLibrary. If the '*IsCameraDirty*' is set to true, then the parameter is passed to the Material Object, as well as setting the view vector, and the camera position parameters within the Material object. These are also retrieved by the static Boolean properties held and set within PVSCameraLibrary of *EyeVector* and *EyePosition* respectively.

Additional research is needed to determine whether this approach is appropriate and also the fastest in terms of processing and rendering speeds. We assume that the processing of setting three static members with one update call to the currently employed 3D camera, if the properties change, only then update the PVSMaterial object, will be faster than extracting the view vector and view position within the IASF in either the Vertex Shader function or Pixel Shader.

8.1.4.2 Shader World Dirty Hook-up

If a PVSNode has moved by translation, rotation, or scale, the lighting calculations are needed to be updated. The PVSNode has a Boolean value '*IsWorldTransformCorrect'*, which is checked and set on each frame. If the PVSNode has translated, rotated, scaled from the previous frame, the 'IsWorldTransformCorrect' is set to false. On the update of the '*ShaderWorldDirtyHookUp*' controller, it checks this Boolean value, and if the attached PVSNode has been translated, the controller will pass the message to the PVSMaterial object, stating to update the lighting calculations of the model. This will recalculate the WorldViewProjection matrix, World matrix, and EyePosition, and pass them to the IASF effect object.

Using this controller as a default releases the need for a direct connection between the PVSSpatial and the PVSMaterial objects of each model part. Not all PVSSpatial objects have an attached PVSModel, hence do not need the attachment of this type of controller, saving processing where not needed.

8.2 PVSModel

The PVSModel object implemented as the specification and design sections states. It contains a list of model parts, which in turn contains the PVSMaterial and IASF effect reference. The PVSModel object has additional functions to allow data dispersal between the nodes of the scenegraph structure, and the model parts. The functions are omitted from the class diagram shown in Figure 97.

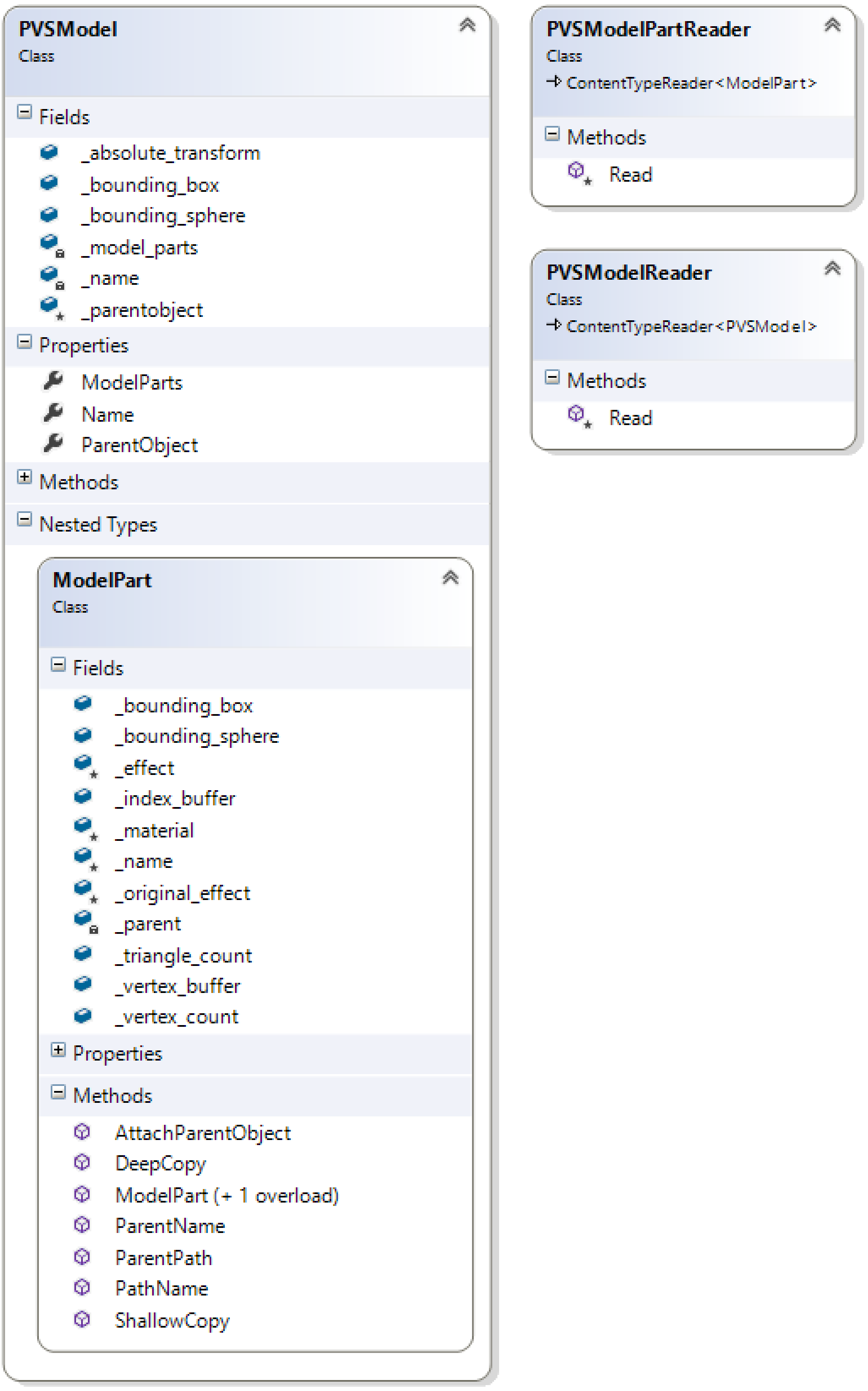
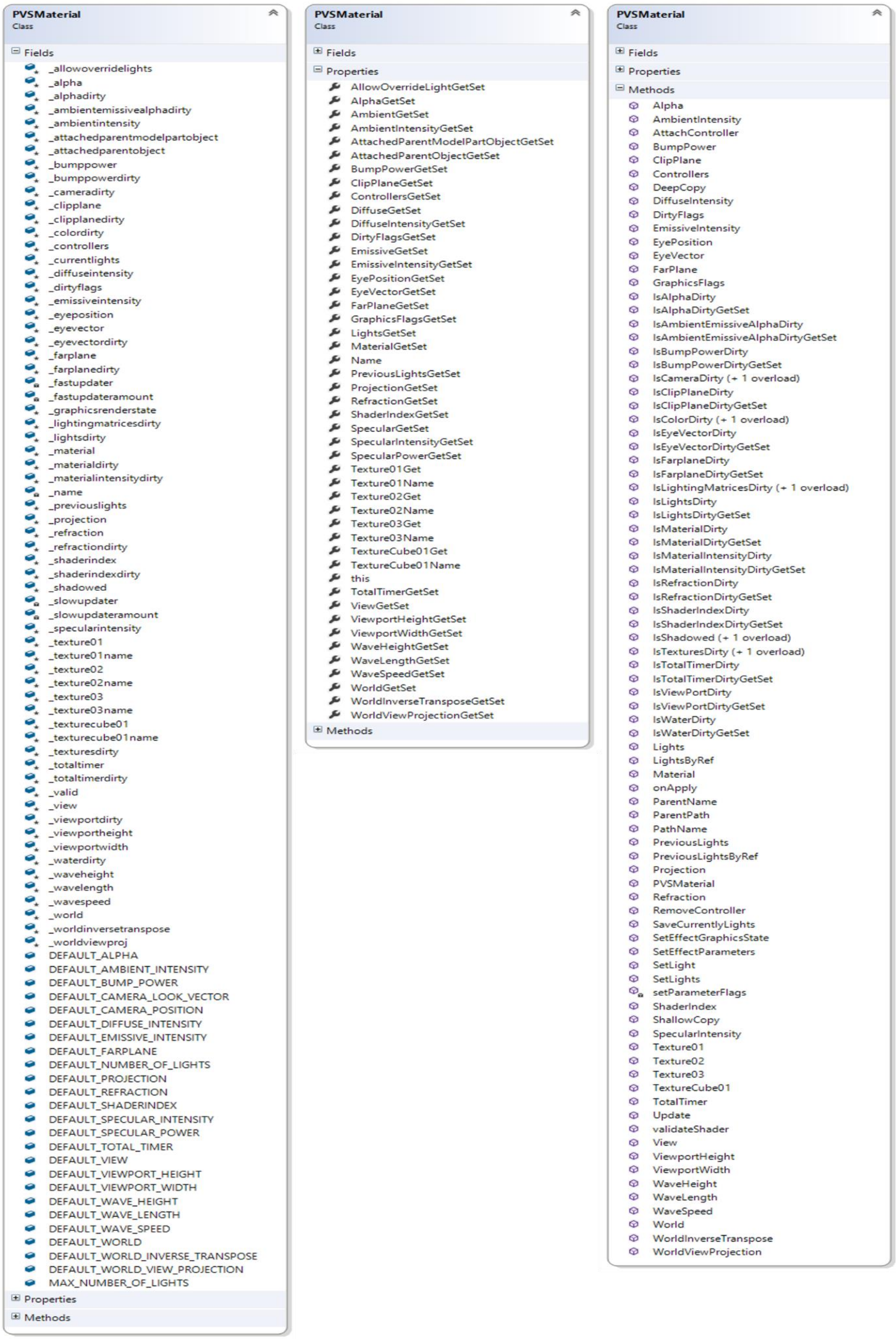


Figure 97 PVSModel and model part class diagram

8.3 PVSMaterial

The PVSMaterial class diagram is shown in Figure 98. The class is large, and thusly the image shows the fields to the left, properties in the middles, and the methods to the right. The PVSMaterial contains information needed for all lighting and effect techniques which the IASF is capable of. Addition of more shading techniques or effects may incur the need to update the PVSMaterial class. The class has been designed to contain data needed for a majority of common lighting and shading techniques.

PVSMaterialDescription class shown in Figure 99 is used to quickly set the basic parameters needed for basic rendering techniques. The PVSMaterialDescription contains the ambient, diffuse, emissive, specular colours, and the specular power. The PVSMaterialDescription is duplicated within the IASF effect HLSL file, thusly the data can be passed to the effect file. Before data is uploaded to the IASF effect file, it passes through an optimiser as depicted in Figure 100. Figure 100 shows the flow of data from the model parts PVSMaterial, updated by the optimiser, to the effect to render the model part. The optimiser is responsible for updating only the properties which need to be updating, as well as compressing data, and removing unneeded data. For example, the ambient and emissive colour values can be combined CPU side and passes as a single colour. The outcome is the same. Only the sum of three Vector4 objects occurs for a model part, instead of summing the value for each vertex or each pixel. We discuss further the HLSL effect object and the optimiser next.


PVSMaterial
Class

⊟ Fields
- _allowoverridelights
- _alpha
- _alphadirty
- _ambientemissivealphadirty
- _ambientintensity
- _attachedparentmodelpartobject
- _attachedparentobject
- _bumppower
- _bumppowerdirty
- _cameradirty
- _clipplane
- _clipplanedirty
- _colordirty
- _controllers
- _currentlights
- _diffuseintensity
- _dirtyflags
- _emissiveintensity
- _eyeposition
- _eyevector
- _eyevectordirty
- _farplane
- _farplanedirty
- _fastupdater
- _fastupdateramount
- _graphicsrenderstate
- _lightingmatricesdirty
- _lightsdirty
- _material
- _materialdirty
- _materialintensitydirty
- _name
- _previouslights
- _projection
- _refraction
- _refractiondirty
- _shaderindex
- _shaderindexdirty
- _shadowed
- _slowupdater
- _slowupdateramount
- _specularintensity
- _texture01
- _texture01name
- _texture02
- _texture02name
- _texture03
- _texture03name
- _texturecube01
- _texturecube01name
- _texturesdirty
- _totaltimer
- _totaltimerdirty
- _valid
- _view
- _viewportdirty
- _viewportheight
- _viewportwidth
- _waterdirty
- _waveheight
- _wavelength
- _wavespeed
- _world
- _worldinversetranspose
- _worldviewproj
- DEFAULT_ALPHA
- DEFAULT_AMBIENT_INTENSITY
- DEFAULT_BUMP_POWER
- DEFAULT_CAMERA_LOOK_VECTOR
- DEFAULT_CAMERA_POSITION
- DEFAULT_DIFFUSE_INTENSITY
- DEFAULT_EMISSIVE_INTENSITY
- DEFAULT_FARPLANE
- DEFAULT_NUMBER_OF_LIGHTS
- DEFAULT_PROJECTION
- DEFAULT_REFRACTION
- DEFAULT_SHADERINDEX
- DEFAULT_SPECULAR_INTENSITY
- DEFAULT_SPECULAR_POWER
- DEFAULT_TOTAL_TIMER
- DEFAULT_VIEW
- DEFAULT_VIEWPORT_HEIGHT
- DEFAULT_VIEWPORT_WIDTH
- DEFAULT_WAVE_HEIGHT
- DEFAULT_WAVE_LENGTH
- DEFAULT_WAVE_SPEED
- DEFAULT_WORLD
- DEFAULT_WORLD_INVERSE_TRANSPOSE
- DEFAULT_WORLD_VIEW_PROJECTION
- MAX_NUMBER_OF_LIGHTS

⊞ Properties
⊞ Methods

PVSMaterial
Class

⊞ Fields
⊟ Properties
- AllowOverrideLightGetSet
- AlphaGetSet
- AmbientGetSet
- AmbientIntensityGetSet
- AttachedParentModelPartObjectGetSet
- AttachedParentObjectGetSet
- BumpPowerGetSet
- ClipPlaneGetSet
- ControllersGetSet
- DiffuseGetSet
- DiffuseIntensityGetSet
- DirtyFlagsGetSet
- EmissiveGetSet
- EmissiveIntensityGetSet
- EyePositionGetSet
- EyeVectorGetSet
- FarPlaneGetSet
- GraphicsFlagsGetSet
- LightsGetSet
- MaterialGetSet
- Name
- PreviousLightsGetSet
- ProjectionGetSet
- RefractionGetSet
- ShaderIndexGetSet
- SpecularGetSet
- SpecularIntensityGetSet
- SpecularPowerGetSet
- Texture01Get
- Texture01Name
- Texture02Get
- Texture02Name
- Texture03Get
- Texture03Name
- TextureCube01Get
- TextureCube01Name
- this
- TotalTimerGetSet
- ViewGetSet
- ViewportHeightGetSet
- ViewportWidthGetSet
- WaveHeightGetSet
- WaveLengthGetSet
- WaveSpeedGetSet
- WorldGetSet
- WorldInverseTransposeGetSet
- WorldViewProjectionGetSet

⊞ Methods

PVSMaterial
Class

⊞ Fields
⊞ Properties
⊟ Methods
- Alpha
- AmbientIntensity
- AttachController
- BumpPower
- ClipPlane
- Controllers
- DeepCopy
- DiffuseIntensity
- DirtyFlags
- EmissiveIntensity
- EyePosition
- EyeVector
- FarPlane
- GraphicsFlags
- IsAlphaDirty
- IsAlphaDirtyGetSet
- IsAmbientEmissiveAlphaDirty
- IsAmbientEmissiveAlphaDirtyGetSet
- IsBumpPowerDirty
- IsBumpPowerDirtyGetSet
- IsCameraDirty (+ 1 overload)
- IsClipPlaneDirty
- IsClipPlaneDirtyGetSet
- IsColorDirty (+ 1 overload)
- IsEyeVectorDirty
- IsEyeVectorDirtyGetSet
- IsFarplaneDirty
- IsFarplaneDirtyGetSet
- IsLightingMatricesDirty (+ 1 overload)
- IsLightsDirty
- IsLightsDirtyGetSet
- IsMaterialDirty
- IsMaterialDirtyGetSet
- IsMaterialIntensityDirty
- IsMaterialIntensityDirtyGetSet
- IsRefractionDirty
- IsRefractionDirtyGetSet
- IsShaderIndexDirty
- IsShaderIndexDirtyGetSet
- IsShadowed (+ 1 overload)
- IsTexturesDirty (+ 1 overload)
- IsTotalTimerDirty
- IsTotalTimerDirtyGetSet
- IsViewPortDirty
- IsViewPortDirtyGetSet
- IsWaterDirty
- IsWaterDirtyGetSet
- Lights
- LightsByRef
- Material
- onApply
- ParentName
- ParentPath
- PathName
- PreviousLights
- PreviousLightsByRef
- Projection
- PVSMaterial
- Refraction
- RemoveController
- SaveCurrentlyLights
- SetEffectGraphicsState
- SetEffectParameters
- SetLight
- SetLights
- setParameterFlags
- ShaderIndex
- ShallowCopy
- SpecularIntensity
- Texture01
- Texture02
- Texture03
- TextureCube01
- TotalTimer
- Update
- validateShader
- View
- ViewportHeight
- ViewportWidth
- WaveHeight
- WaveLength
- WaveSpeed
- World
- WorldInverseTranspose
- WorldViewProjection

Figure 98 PVSMaterial class diagram. Left is fields, middle is properties, and right are methods.

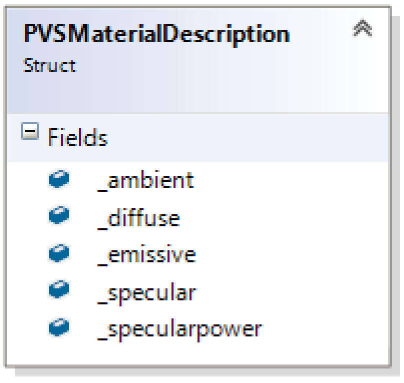


Figure 99 PVSMaterialDescription structure. Used to group material parameters commonly used on model mesh rendering.

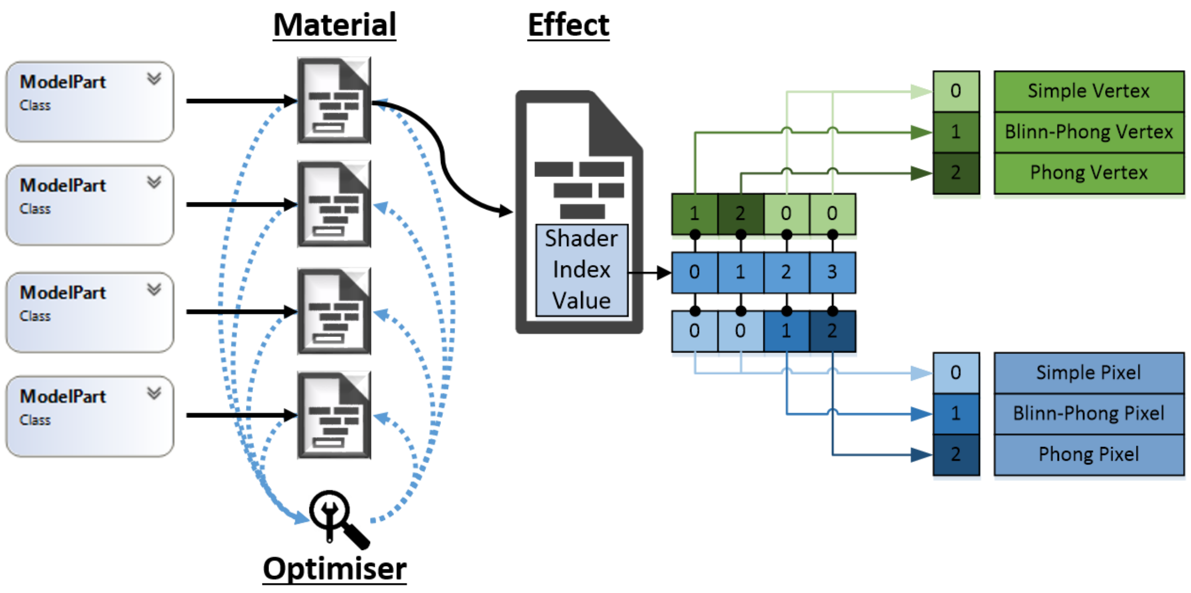


Figure 100 Material data flow from model part material, through the optimiser, to the effect object for rendering.

8.4 HLSL Effect Object and Effect Optimiser

We have utilised the HLSL effect object used for the XNA API. The effect file is to be used for all objects within a scene for rendering which reduces the effect count of; many to 1 instance. The effect file contains all the parameters, properties, functions, and structures needed to implement multiple rendering techniques. As stated, the name of this type of effect file is called IASF. Models being rendered using the effect file need the vertex data structures to contain correct data; position, UV coordinate, binormal, and tangent value.

The effect optimiser tracks which effect parameters need to be recomputed during the OnApply function call. It utilised flags in the form of an enumeration. The enumeration type allows for the testing of the values in switches and if-statements. They are used as bit-fields, bit 1 is a flag which states the WorldViewProjection matrix needs updating, whereas bit 8 states the material colour needs to be recomputed. The enumeration values need to have the values assigned.

The enumeration representations are shown in Table 16.

Name	Value	Bit Representation
WorldViewProj	1	1
World	2	10
EyePosition	4	100
MaterialColor	8	1000
Textures	16	10000
ShaderIndex	32	100000
EyeVector	64	1000000
ViewPort	128	10000000
Material	256	100000000
Lights	512	1000000000
MaterialIntensity	1024	10000000000
ClipPlane	2048	100000000000
Alpha	4096	1000000000000
Bumppower	8192	10000000000000
TotalTimer	16384	100000000000000
Water	32768	1000000000000000
Farplane	65536	10000000000000000
AmbientEmissiveAlpha	131072	100000000000000000
All	-1	-1

Table 16 Effect dirty flag enumeration representation.

The enumeration is instantiated in the PVSMaterial object. Within the update of the PVSNode, and the material controllers, various properties may be modified. Within the setter functions, the property is updated, and the corresponding flag is modified to be declared as 'dirty'. For example, setting a new diffuse colour will set the MaterialColor flag to 1, and the translation of the model will set the WorldViewProj, and World flags will be set to 1. This gives; 1011.

The combination of dirt flags are sent to the effect helper and checked. If a flag is set as dirty, the parameters are recalculated, and uploaded to the GPU. The binary flags are removed from the dirty flag enumeration. The checks are completed by utilised binary bitwise operations. To check if the flag has been set, the dirty flag is compared using the single bitwise AND operator. If the value is not equal to 0 then the recalculation and upload to GPU will commence. After recalculation, the dirty flags variable will have the current flag removed using the bitwise complement.

We utilise this technique to group commonly changing properties such as the WorldViewProjection matrix along with the separate World, View, and Projection matrices together. They are then recalculated together, optimising the calculations to only be done when needed.

The other parameters which are passed through to the effect file are shown in Table 17. We also state the intentions for the parameters.

Datatype	Name	Description
Matrix 4x4	World	World matrix of the model
	View	View matrix of the camera
	Projection	Projection matrix of the camera
	WorldViewProj	Combined matrix of the World, View and Projection matrices.
	WorldInverseTranspose	Inverse world matrix.
Vector4	ClipPlane	Used for depth tests.
Vector3	EyeVector	View direction.
	EyePosition	The 3D position of the camera.
Float	Ambient Intensity	Intensity value between 0-1.
	Diffuse Intensity	Intensity value between 0-1.
	Specular Intensity	Intensity value between 0-1.
	Emissive Intensity	Intensity value between 0-1.
	Alpha	Alpha value between 0-1.
	Bump Power	Intensity of the normal-mapping.
	Total Timer	Total simulation passed time.
	Wave length	The length of wave used for water rendering.
	Wave Height	The height value of the wave for water rendering.
	Wave Speed	The speed value of the wave for water rendering.
	FarPlane	Distance to farplane.
Int	ShaderIndex	The index value for choosing the rendering technique.
	Viewport Width	Width of the viewport. Used for occlusion techniques.
	Viewport Height	Height of the viewport. Used for occlusion techniques.
	Mip Level	Used for sampling of mip-mapped textures. [80]
Material Structure	_Material	Material structure used for storing the ambient colour and others.
Light Structure	Lights[n]	An array of light structures used to store the multiple lights applied to a scene.

Table 17 Properties within the effect file.

8.4.1 Texture rendering for unified Terrains

Graphical programmers often do not combine per-vertex rendering with textures. When rendering a model mesh with a texture, the texture is sampled for the given pixel, so is often used with per-pixel rendering. It is uncommon enough that some GPUs do not allow this functionality of sampling a texture within the vertex function of the effect file, and will often default to a pre-defined colour; normally black. We stated that this is a large issue, especially with rendering terrains utilising real-world map textures. We state that utilising per-vertex shading sampling textures within the vertex function is needed for the instance that a model mesh can be the same width and height of a texture, i.e. terrain maps are often square, and textures can be placed on top of that model mesh. If the texture is the same width and height in pixels as the terrain is with vertices, using per-vertex texture sampling will be less processing than that of per-pixel, but with the same result.

8.4.2 Vertex Texture Fetch

Stated in the specification, the IASF will contain a configuration which uses per-vertex Blinn-Phong and per-vertex Phong shading with 'vertex texture fetch', which is available within shader model 3, and with certain NVIDEA graphics cards.

The keyword is 'tex2Dlod'[60,61] is used to extract texture information within a vertex shader function. This is limited to the pixel location, which is suitable for placing textures onto model mesh object with the same width and height of vertices as the textures pixel. The terrain data and terrain texture overlays are of equal parameters, and this technique is suitable.

Although this is possible, it would be unjust due to the increased processing needed; which negates the use of an IASF. We reserve the need to use this technique in special case circumstances.

8.5 Indexed Array Shader Function Implementation

This section discusses how the IASF has been implemented within the PVS framework. This is how IASFs are intended to be implemented, but our version has supporting classes to streamline development and increase performance.

Table 18 shows the bit-field representation used for combining the varying rendering techniques which can be combined and applied to a model mesh. Table 18 also contains 'Special' instances of techniques; flat shading, and texture only. These techniques cannot be combined with other techniques. These types of special values can be referenced bit values higher than needed for the techniques which can be combined. Table 18 shows that per-vertex and Blinn-Phong both have the

[60] https://msdn.microsoft.com/en-us/library/windows/desktop/bb509680(v=vs.85).aspx
[61] http://http.developer.nvidia.com/Cg/tex2Dlod.html

same bit value which is 0 in decimal. This is because all model mesh objects will be rendered as default per-vertex Blinn-Phong shaded. Combining other techniques (Diffuse mapped, Normal mapped, Specular mapped, reflection, refraction, hemispherical) with per-pixel rendering will only need the addition of ‘2’ added to the shader index. To improve rendering quality and realism, but increase processing, adding ‘1’ will shade objects using the Phong lighting equation instead of Blinn-Phong equation. This proves interesting because, if framerates are below an acceptable value, then selected assets can negate a value of 1 from the shader index and be rendered using the Blinn-Phong lighting calculation instead of Phong (if Phong lighting was already selected). Reducing calculations further, negating 2 from the shader index will convert the models rendering from per-pixel to per-vertex. This is a form of LoD. There are 255 combinations of shading effect we can utilise; this does not include the use of different lights.

Table 19 shows a subset of the index value and the corresponding lighting calculations and rendering techniques used. With the large amount of combinations, we have not displayed them all.

Technique	Bit Value	Binary Representation
Per-Vertex	0	0000000000000000
Blinn-Phong	0	0000000000000000
Phong	1	000000000000000**1**
Per-pixel	2	00000000000000**1**0
Diffuse Mapped	4	0000000000000**1**00
Normal Mapped	8	000000000000**1**000
Specular Mapped	16	00000000000**1**0000
Reflection	32	0000000000**1**00000
Refraction	64	000000000**1**000000
Hemispherical	128	00000000**1**0000000
Light 1 A	0 * 256 = 0	0000000000000000
Light 1 P	1 * 256 = 256	0000000**1**00000000
Light 1 A Light 2 A	2 * 256 = 512	000000**1**000000000
Light 1 A Light 2 P	3 * 256 = 768	000000**11**00000000
Light 1 P Light 2 A	4 * 256 = 1024	00000**1**0000000000
Light 1 A Light 2 A Light 3 A	5 * 256 = 1280	00000**1**0**1**00000000
Light 1 A Light 2 A Light 3 P	6 * 256 = 1536	00000**11**000000000
Light 1 A Light 2 P Light 3 P	7 * 256 = 1792	00000**111**00000000
Light 1 P Light 2 P Light 3 P	8 * 256 = 2048	0000**1**00000000000
No Lights	Special	User defined
Flat Shader	Special	User defined
Texture Only	Special	User defined
Environment Mapped	Special	User Defined

Table 18 Bit-field layout structure for our Indexed Array Shader Function. Special values are for unique rendering techniques which cannot be combined with other techniques.

Shader Index Value	Vertex or Pixel Shaded	Lighting Calculation	Diffuse Textured	Binary
0	Per-Vertex	Blinn-Phong		0000
1	Per-Vertex	Phong		0001
2	Per-Pixel	Blinn-Phong		0010
3	Per-Pixel	Phong		0011
4	Per-Vertex	Blinn-Phong	True	0100
5	Per-Vertex	Phong	True	0101
6	Per-Pixel	Blinn-Phong	True	0110
7	Per-Pixel	Phong	True	0111

Table 19 Shader index subset representation showing index value and correspond rendering technique and binary representation.

8.5.1 Indexed Array Shader Function Index Selection

Within this section we show the technique we will use to determine the shader index needed for rendering objects by a specific shading technique.

Due to the size of the number of permutations of shader we need to develop, maintain, and adapt, we will focus on the most common permutations of techniques; per-vertex, per-pixel, Blinn-Phong, Phong, texture 01, texture 02, texture 03, reflection, refraction, and hemispherical. Texture 01, 02, and 03 represent diffuse, normal, and specular texturing. This section will not state all the permutations of the techniques shown in Table 18. but introduce the technique for combining the techniques to generate the permutations.

To represent the technique to use, a bit-field will provide the flexibility needed to create the number of permutations needed; 255 with initial estimates. The bit-field is used to encode states of rendering. Per-vertex is a state, while per-pixel is another state. Textured is a state, while non-textured or normal textured are other states. These can be used together or separately; for which a bit-field is best suited.

To encode 10 functions, 8 bits will be needed. The bits we are representing are as stated in Table 20.

Technique	Binary Representation	Index
Per-Vertex	00000000	0
Blinn-Phong	00000000	0
Phong	00000001	1
Per-Pixel	00000010	2
Diffuse Textured	00000100	4
Normal Textured	00001000	8
Specular Textured	00010000	16
Reflection	00100000	32
Refraction	01000000	64
Hemispherical	10000000	128

Table 20 Technique and Binary representations with shader indexes for use with the Indexed Array Shader Function.

The reason why we use 0 for per-vertex and Blinn-Phong is because this will be the default rendering technique for all models. Offsetting the Per-pixel techniques to bit 2 instead of laying it next to related per-vertex function allows for equal offsetting of techniques between per-vertex and per-pixel rendering.

Table 21 displaying the per-vertex and per-pixel layouts from 0 to 31. Each value between these represent a technique, or a combination of techniques. There are 255 combinations for the techniques shown in Table 20. Additional values are needed for custom techniques; rim-lighting, post-processing.

Table 21 shows the 31 permutations and the binary representations they are encoded as. During the implementation phase, the rest of the permutations will be encoded.

The next section shows how we intend to encode the light structures for improved rendering, and to minimise the branching within the GPU.

Shader Index	Shader Function	Lighting Model	Diffuse Texture	Normal Texture	Specular Texture	Binary Representation
0	Per-Vertex	Blinn-Phong				00000000
1	Per-Vertex	Phong				0000000**1**
2	Per-Pixel	Blinn-Phong				000000**1**0
3	Per-Pixel	Phong				000000**11**
4	Per-Vertex	Blinn-Phong	Yes			00000**1**00
5	Per-Vertex	Phong	Yes			00000**1**0**1**
6	Per-Pixel	Blinn-Phong	Yes			00000**11**0
7	Per-Pixel	Phong	Yes			00000**111**
8	Per-Vertex	Blinn-Phong		Yes		0000**1**000
9	Per-Vertex	Phong		Yes		0000**1**00**1**
10	Per-Pixel	Blinn-Phong		Yes		0000**1**0**1**0
11	Per-Pixel	Phong		Yes		0000**1**0**11**
12	Per-Vertex	Blinn-Phong	Yes	Yes		00001100
13	Per-Vertex	Phong	Yes	Yes		00001101
14	Per-Pixel	Blinn-Phong	Yes	Yes		00001110
15	Per-Pixel	Phong	Yes	Yes		00001111
16	Per-Vertex	Blinn-Phong			Yes	00010000
17	Per-Vertex	Phong			Yes	00010001
18	Per-Pixel	Blinn-Phong			Yes	00010010
19	Per-Pixel	Phong			Yes	00010011
20	Per-Vertex	Blinn-Phong	Yes		Yes	00010100
21	Per-Vertex	Phong	Yes		Yes	00010101
22	Per-Pixel	Blinn-Phong	Yes		Yes	00010110
23	Per-Pixel	Phong	Yes		Yes	00010111
24	Per-Vertex	Blinn-Phong		Yes	Yes	00011000
25	Per-Vertex	Phong		Yes	Yes	00011001
26	Per-Pixel	Blinn-Phong		Yes	Yes	00011010
27	Per-Pixel	Phong		Yes	Yes	00011011
28	Per-Vertex	Blinn-Phong	Yes	Yes	Yes	00011100
29	Per-Vertex	Phong	Yes	Yes	Yes	00011101
30	Per-Pixel	Blinn-Phong	Yes	Yes	Yes	00011110
31	Per-Pixel	Phong	Yes	Yes	Yes	00011111

Table 21 Shader Index permutation combination and binary representations

8.5.2 Encoding Lights into the Indexed Array Shader Function

Section 8.5.1 states the encoding of rendering permutations using bit-fields. We have not stated the design to encode lights into the IASF, but we have specified within the specification section that the framework will utilise forward rendering with three lights.

The use of lights within the scenes can be encoded into the permutations by offsetting the values of the encoded rendering techniques by the total number of encoded techniques. To optimise and restrain the use and design of the IASF with respect to the lights passed from the PVSMaterial, the order of the lights is important. We have stated the two lights are needed; ambient and point. An algorithm is needed to organise the combinations of these lights. The organisation of the lights will put the ambient lights before the point light. This will create a unified offset which is used to determine the rendering technique and functions chosen. Figure 101 shows the IASF techniques to the left, and the light combinations in the middle which provides the offset. The algorithm is needed within the material object or the material helper, and will organise the lights when passing to the GPU, or when a light is modified.

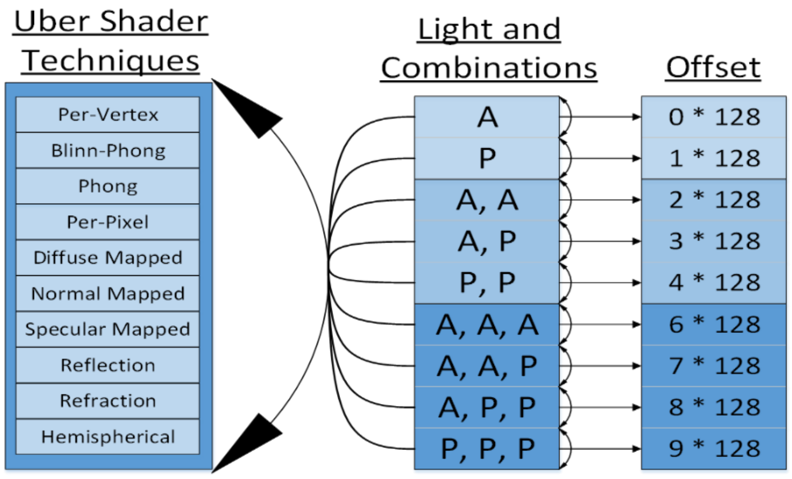


Figure 101 Light encoding offset description. Left is the Indexed Array Shader Function techniques, right is the offsets of the light types and combinations. A is ambient light type, P is point light type.

8.6 GPU Render State Management Implementation

To generate the large varying amounts of render states used for multiple and permutated rendering shading techniques we utilise bit-fields similar to the technique used for selecting the IASF functions to use stated in section 8.5. The IASF selection technique accounts for a majority of permutations and combinations of the commonly used rendering techniques.

As stated in the design section, four techniques can be employed when changing the state of the GPU for rendering model meshes. We iterate that the choice to choose option 4 is valid. The technique is to allow model objects to only change the state of the GPU which is different from the previous render.

The bit-fields are shown in Table 22.

As stated, the generation of pre-defined enumerations are easily generated. This proves well for future additions and combinations of render state selection.

Blend Mode	Value	Binary
Opaque	0	0000,0000
Translucent	1	0000,0001
Additive	2	0000,0010
NonePremultipliedAlpha	4	0000,0011
CullNone	8	0000,0000
Cull CounterClockwise	16	0000,0100
CullClockwise	32	0000,1000
Solid	64	0000,0000
Wireframe	128	0001,0000
EnableDepth	256	0000,0000
DisableDepth	512	0010,0000
DepthTestButNoWrites	1024	0100,0000
StencilShadowMode	2048	1000,0000

Table 22 Graphics render state calculator bit-field representation.

8.6.1 Pre-built render state functions

To allow users to select a rendering technique and accompanying set the GPU render state concurrently without the knowledge of setting individual parameters needed, we create pre-selected rendering options for a user to choose from. This includes default GPU render states, and some pre-made render state collections used for transparency, and other techniques. The functions are split into two varying techniques; some functions create the GPU render state '*BlendState'* function and return said GPU parameter, while others create a bit-field representation which then selects the individual options within the '*SetRenderStates'* function within the '*GraphicsStateHelper'* class.

The pre-built GPU render states can be dispersed through the scenegraph to change the visuals of assets of single or multiple objects.

8.7 Interactive Visualisation Interface

To aid development, and allow users to utilise the framework coherently, the development and implementation of an IVI system is needed.

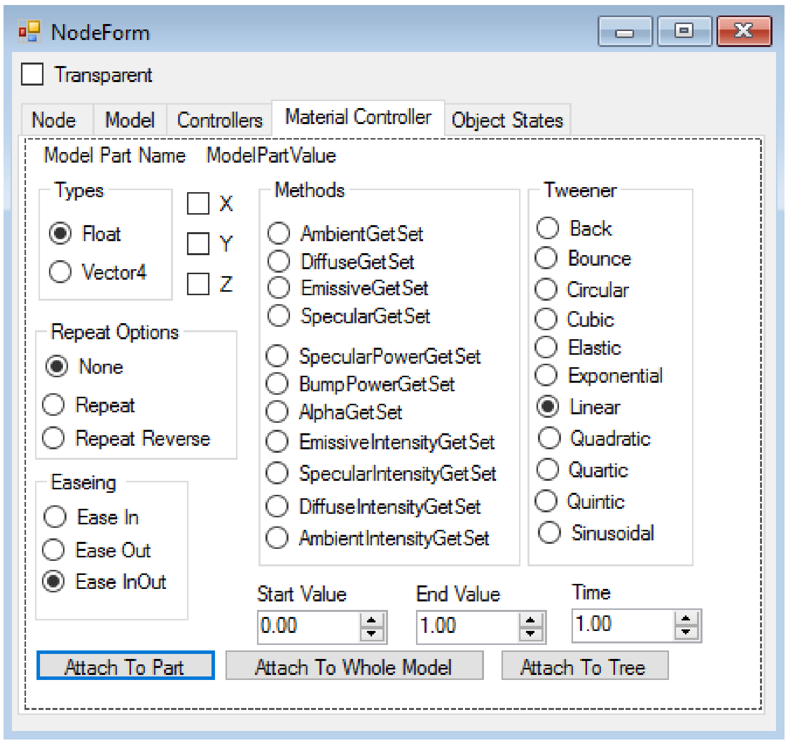
The IVI system allows the interaction with the main aspects of the framework; the scenegraph and access to individual nodes, the global light manager, search capabilities upon the scenegraph, user generated query searches, location trackers, access to models within nodes and their model parts, access to the material of model parts to allow modification to GPU properties, and the ability to generate controllers which can be attached to properties, or individual components of properties i.e. the XYZW components of a Vector4 object. We will discuss the development of the controller generation, the GPU render state generation, and the shader index value calculator.

8.7.1 Controller Generation

We have two types of controller to generate; spatial and material. We will depict the creation of a material controller as to include the use of the Tweener object which is used to animate the object or property it is attached to.

Properties which of the Material object are either a float, or a float component of a Vector4 object. The user can select either. If the Vector4 datatype is chosen, they can choose to which component they wish to apply the animation too. The properties are listed in the middle. The user can choose to make the animation loop from start to finish, or loop from start to finish back to the start again. The multiple functions can be chosen as well from ease-in, ease-out, or ease-in-out, along with the type of function; linear, quadratic, elastic etc. The start value, end value, and the timespan which it will animate through. The controller can be attached to a single part, the whole model, or applied to all nodes of the subtree which are below the selected node.

This allows for a large number of animation controllers to be applied to many of the properties of the material object.


NodeForm

☐ Transparent

Node | Model | Controllers | Material Controller | Object States

Model Part Name ModelPartValue

Types: [✓ Float] Vector4

☐ X ☐ Y ☐ Z

Methods: AmbientGetSet, DiffuseGetSet, EmissiveGetSet, SpecularGetSet, SpecularPowerGetSet, BumpPowerGetSet, AlphaGetSet, EmissiveIntensityGetSet, SpecularIntensityGetSet, DiffuseIntensityGetSet, AmbientIntensityGetSet

Tweener: Back, Bounce, Circular, Cubic, Elastic, Exponential, [✓ Linear], Quadratic, Quartic, Quintic, Sinusoidal

Repeat Options: [✓ None] Repeat, Repeat Reverse

Easeing: Ease In, Ease Out, [✓ Ease InOut]

Start Value 0.00 End Value 1.00 Time 1.00

Attach To Part | Attach To Whole Model | Attach To Tree

Figure 102 Material Controller IVI.

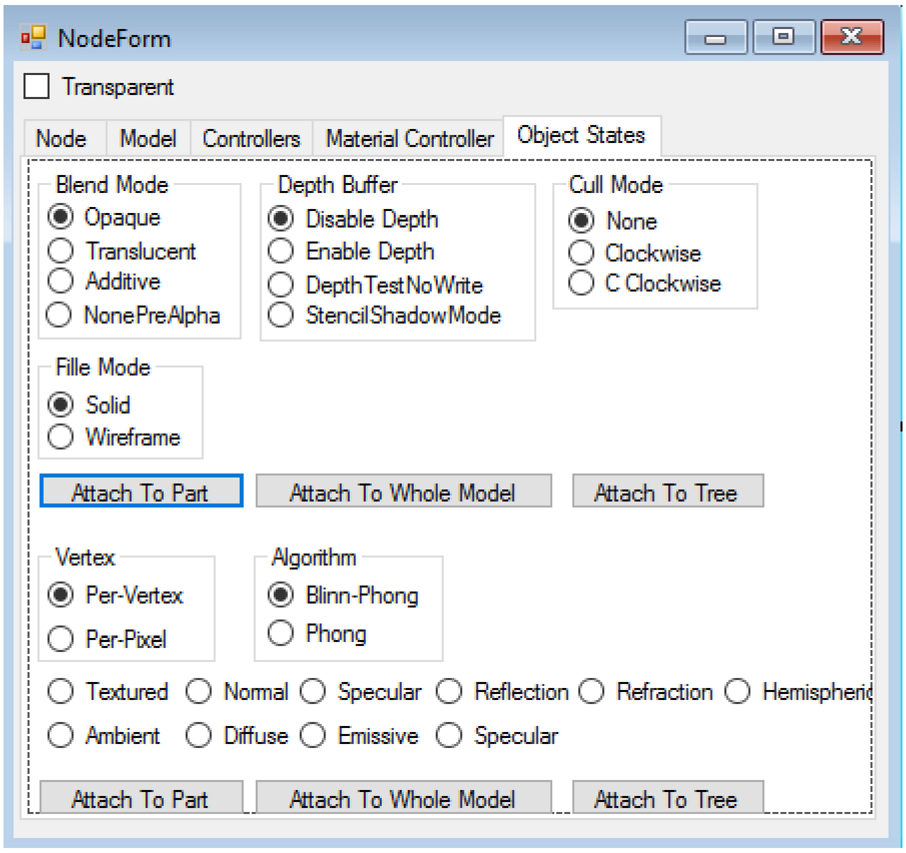
8.7.2 GPU Render State Generation

The GPU render states are generated by checking the subsets of the four subset categories; Blend Mode, Depth Buffer, Cull Mode, and Fill Mode. All values are contained within an enumeration bit-field. The default values are set to zero within the enumeration bit-field flag. The subset categories are shown in Figure 103.

All subsets have a default of zero; Opaque is equal to 0, so is: disable depth, no culling and solid. This allows for the default settings to be quickly generated. When the user applies the GPU render state to either a single model part, or attaching to all model parts of the mode, or to the subtree from this node, the check boxes are added together by retrieving the unsigned integer from the enumeration.

During the rendering, the GPU state which is attached to the material object of the model part is checked against the current GPU render state. If the value is different from the current GPU state, the GPU state is updated. The check between the current GPU state, and the GPU state needed by the model part is done by X-OR operation. If the value is not zero then a render state is different. Each subset is checked by using the operation of AND to check if a component in the subset is not equal to zero. If it is not, then an element has been changed, and will go through a switch case to modify the component which has changed.

This technique allows for a large permutation of render states to be applied to all models, or subsets of models within a scene.


NodeForm

Transparent

Node | Model | Controllers | Material Controller | Object States

Blend Mode: [✓ Opaque] Translucent Additive NonePreAlpha

Depth Buffer: [✓ Disable Depth] Enable Depth DepthTestNoWrite StencilShadowMode

Cull Mode: [✓ None] Clockwise C Clockwise

Fille Mode: [✓ Solid] Wireframe

Attach To Part | Attach To Whole Model | Attach To Tree

Vertex: [✓ Per-Vertex] Per-Pixel

Algorithm: [✓ Blinn-Phong] Phong

Textured Normal Specular Reflection Refraction Hemispheric

Ambient Diffuse Emissive Specular

Attach To Part | Attach To Whole Model | Attach To Tree

Figure 103 GPU render state generation and shader index calculator.

8.7.3 Shader Index Value Calculator

The choosing of the shader index which in turn chooses the vertex and pixel shader function is also shown in Figure 103. The shader index design is shown in section 8.5. The check boxes contained within the boxes; per-vertex, per-pixel, Blinn-Phong, and Phong are housed together because the design of the IASF uses these values as offsets. The per-vertex and per-pixel cannot be both chosen, as with Blinn-Phong and Phong. Texture mapping as will the disuse, normal, and specular maps are normally done with per-pixel shading, but as stated in the specification and design section, a user may wish to use per-vertex texturing; the model vertices must align with the texture correctly for this to be a suitable option for rendering. This does allow for greater flexibility, but must adhere to the knowledge and understanding on how the rendering pipeline works. The other elements; textured, normal mapped, specular mapped etc. can be combined in any order. This IVI allows the modification of the shader index to a single model part, to a whole model, or a subset of the scenegraph. This allows great flexibility with specifying the shading parameters of all models within a scene.

8.8 Conclusion

We conclude the current framework satisfies the specification needs stated throughout this work. The pre-processing pipeline allows the pre-processing of the data sets available, producing outputs for use not only within the runtime simulation but within alternative frameworks and projects. The pipeline generates data where there was not data, and uses the base and inferred data to generate procedural model mesh objects for the visualisation within the runtime environment. To organise the data, a scenegraph is implemented. The novel scenegraph structure allows the adaptation, organisation, and visualisation of large dense urban environments, while allow dispersal of user commands and runtime data structures. Data which is passed through the scenegraph can be used for setting material parameter within the custom model object. The material will optimise the property settings, calculating results CPU side, before passing them to the IASF effect model on the GPU. The IASF utilised with the custom model mesh object allows improved rendering speeds through the reduction of HLSL branching, and flexibility with shader indexing allow all model objects to be rendered with any combination of lighting and rendering techniques.

To determine if the project and framework is successful, the evaluation of the framework is needed. The evaluation is discussed next.

9 Evaluation of Big Geospatial Data Framework

Within this chapter, we evaluate the framework in regards to the evaluations stated within section 1.2. We remind the reader that the evaluations will include; recording the frame rate of the runtime simulation, the update time during a frame, as well as the rendering time of a frame and GPU render time. The processing time of selected algorithms will also be presented and evaluated.

The visual appearance of a scene will also be critiqued, and compared against similar GIS applications, frameworks, and platforms. We will compare against a single platform, OSM2World for which we have obtained source code. This allows the comparison of many parameters; processing of scenes, and visual appearances.

The processing, and traversal search time will also be recorded from user-generated queries.

The dispersal of data through the scenegraph will also be recorded.

The features and components of the framework will be critiqued and compared against; Google Maps[62], Arc GIS[63], OSM[64], SAVE[1], ALLADIN[5]–[7], and the work of Bo Mao[11], [25].

To evaluate our framework further, we compare the performance of the simulation on multiple machines, a low-range PC, mid-range laptop, and a high-end gaming laptop. The main differences between the systems are the CPU, and GPU. This will produce a scalability chart against scenes of various sizes, and densities.

We start by stating the hardware configurations used for the comparison scalability evaluations.

9.1 Experiments' Hardware Configurations

We will evaluate the system on a number of computer configurations. We have access to a number of hardware ranging from a low range PC, a mid-range laptop, and a high-end gaming laptop.

Low Range PC – a 64bit Windows7 Operating System, 4GB of RAM, Intel Core i3-2120 CPU @ 3.3GHz, Intel HD Graphics Card.

Mid-Range Laptop - a 64bit Windows10 Operating System, 8GB of RAM, Intel Core i7-3630QM 4 (2 logical cores per physical) core CPU at 2.40GHz, and an Intel HD Graphics 4000 Chip with an

[62] https://www.google.co.uk/maps
[63] https://www.arcgis.com/
[64] https://www.openstreetmap.org

additional NVIDIA GeForce GTX 660M. The hard-drive of the computer is a Solid State Drive (SSD). An SSD has faster read and write properties than HDD.

High-End Gaming Laptop – an Alienware 15 64bit Windows10 Operating System, 16GB of RAM, Intel Core i7-4710HQ. An additional desktop GPU block is used which is a GTX 980 8GB.

The next section discusses the 5 maps.

9.2 PVS Framework Class and Line count

The PVS Class and Line count can be found within Table 23. This is shown to highlight the various library and development needed for a modern GIS framework. The total number of classes for the framework totals 356. The total number of line totals 20,128. This does not include the rendering API of XNA, nor the .NET framework API.

Library	Line Count	Class/Structure Count
Dynamic Expression Library	874	10
OSM Enum Library	1070	0
PVS Camera Library	372	17
PVS Main	14,386	209
PVS Input Manger Library	101	2
PVS Runtime Library	2,927	97
PVS XML Serialiser	21	1
Triangulator Library	237	7
Tweener Library	140	13
Total	**20,128**	**356**

Table 23 PVS Framework Line and Class count.

9.3 Processing Maps of Varying Size

Table 24 shows the details of the maps we have chosen. The processing time is recorded as well as the number of nodes generated within the novel scenegraph structure. The number of nodes are counted as the nodes serialised to XML and then to XNB files. The number of OSMNodes are recorded within the processor, as well as the number of OSMWays. We record the number of buildings within the OSM files, and check this against the number of buildings generated within the processors. The number of highways is extracted from the OSM file by hand, and checked against the number of highways generated within the processor. We extract the number of highways by hand from the OSM file because we need to remove the number of highway crossings and number of traffic signals. The reason for this is because searching for the text of <tag k="highway" will return

the number of found but include traffic signals, crossings, bus stops, and mini roundabout locations. We will search for these and negate them from the result. The reason we don't search for a type of highway is due to the large number of highway types.

Our first generation of Map 01 led to an issue of a large amount of unneeded scenegraph nodes are being generated. This is due to the resizing of the OSM bounds to include all nodes of OSM, as already stated. The reason this is an issue for this map is that the water boundary and boundary for an area of Liverpool is contained within the OSM file. This results in 27,998 nodes being generated for a small map. A huge number of these nodes are the empty subtrees which we have stated are not needed. A boundary model mesh is created which is the size of Liverpool county, thusly all the tiles and subtrees which this boundary includes are created.

The changing of the boundary size is a justifiable alteration, but a process in placing an asset is to calculate its centroid and use that as the translation to calculate which tiles to generate. Alterations to this calculation are needed. During the check to see which node of the scenegraph to attach the model asset to, if a node does not exist then the model will be removed. This may lead to assets being prematurely removed; but will fix the issue.

The generation time will be evaluated. In this evaluation, we will monitor how long it takes to process the OSM map in terms of creating the model mesh objects, and saving the data into the XML files. The process incurs two parts; the generation and serialisation of XML files (1 for each node of the scenegraph), and then processing the files into formatted XNB binary files used within the runtime simulation. The average processing time cannot be determined either due to the variable file size of the XML file sizes; the model mesh is the major variance in this case.

Table 25 shows the polygon count and the vertex count for each map. Table 26 shows the Latitude and Longitude coordinates of the 5 maps used for the evaluations. Figure 106 shows the map images of the areas used for evaluation.

Map	Scenegraph PVSNode Count	OSMNodes	OSMWays	OSM Building	OSM Highways
1	331	1974	242	32	101
2	631	3520	524	141	219
3	1427	6729	1095	405	452
4	2982	12570	2208	936	845
5	4495	20270	3752	1705	1338

Table 24 OSM Map Details.

Map	Polys	Verts
01	1,992,117	1,041,570
02	1,999,972	1,062,968
03	4,001,002	2,139,886
04	8,987,450	4,740,352
05	NA	NA

Table 25 Map properties at runtime.

Map	MinLat	MinLon	MaxLat	MaxLon
1	53.4040000	-2.9994000	53.4079000	-2.9919000
2	53.4029000	-2.9994000	53.4092000	-2.9876000
3	53.4017000	-2.9994000	53.4109000	-2.9823000
4	53.3995000	-2.9994000	53.4132000	-2.9765000
5	53.3972000	-2.9994000	53.4153000	-2.9680000

Table 26 Map bounds of the 5 maps, used for evaluation.

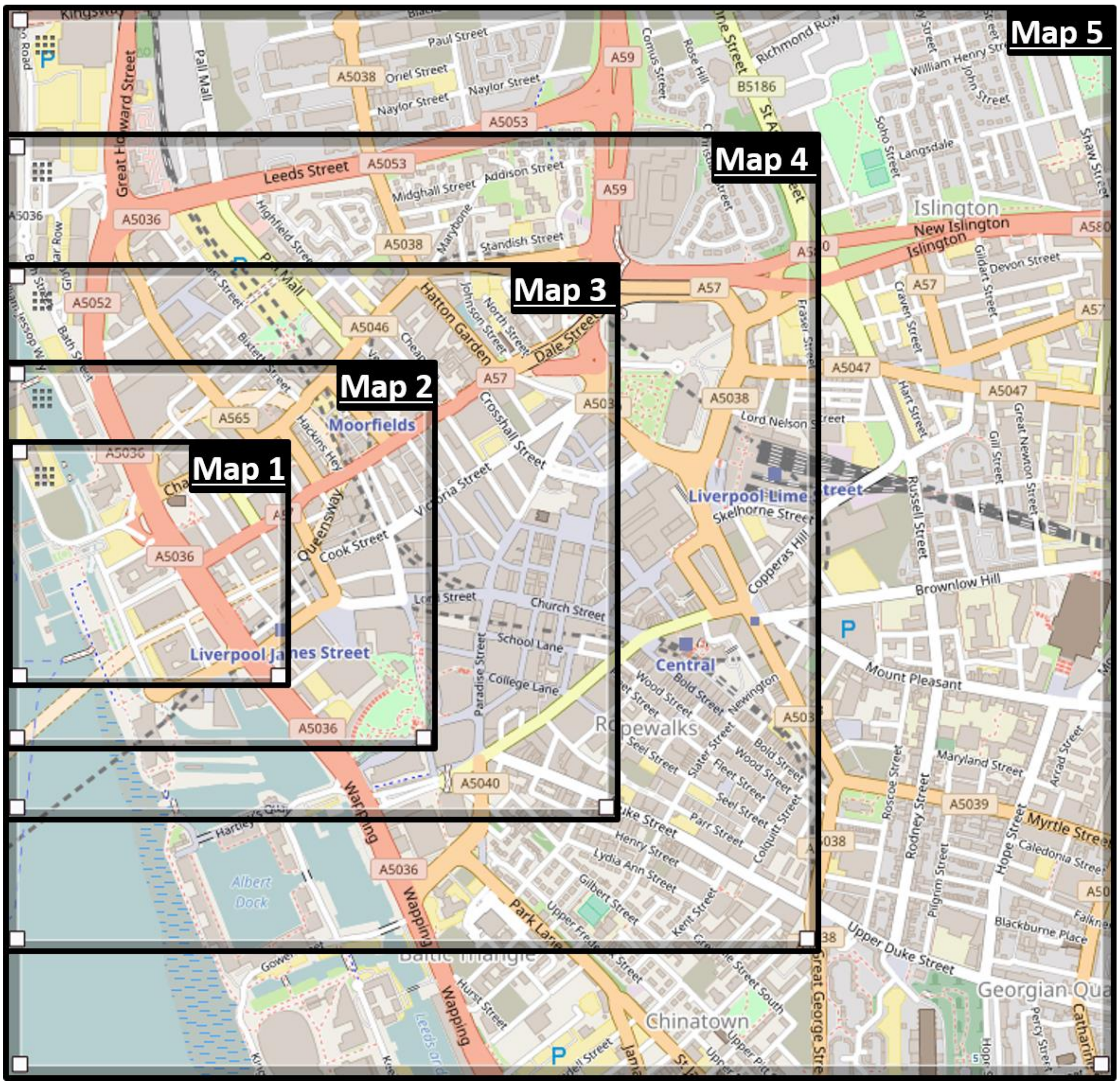


Figure 104 5 OSM Maps used for evaluation overlapped on one another. Images obtained from https://www.openstreetmap.org/search?query=trafford%20centre%20manchester%20uk#map=14/53.4073/-2.9623

9.4 Uniform Scaling of Nodes

Within this experiment, we generate a squared function which duplicates a type of node to monitor the memory usage. The monitoring of 3 types of node will be undertaken; an empty node which does not have a model mesh object applied, a node which contains a procedurally generated building of type 'house', and a user generated 3D model mesh of a typical UK terrace home. The user generated model mesh will contain diffuse textures and be of low to medium polygon count.

The experiment will evaluate the memory usage with a subset of common type objects which are uniformly placed within a geospatial location; a terrace home. This will provide information into the limits of how many buildings of a particular type can be loaded. Within Figure 106, 641 buildings are of type 'house' are shown. This will be used as the scaling format. The area of the chosen location is

0.6km^2. Increasing the average number of buildings to 1km^2 will result in 0.4% increase of number of buildings. This will create an extra 256.4 building objects; resulting in 897.4 buildings of type 'house' per km^2. This is a rough estimate as the nature of geospatial data is not uniform. This can be seen within Figure 106 with part of the map being park land at the top right.

The experiment will compare n^2 objects. *n* will initially be set to 1, and then increased to; 2, 4, 8, 16, and 32. Figure 105 is a visual representation of the uniform scaling procedure.

Before we present the evaluations for the experiment; two issues arose with the underlying architecture of the rendering engine which the framework is built upon; Microsoft XNA. XNA utilises object referencing, so the model mesh object is loaded once but referenced multiple times. This is beneficial when dealing with model mesh objects which are repeated within a scene. This is not the same as hardware instancing[65].

Procedural model mesh objects are created as individual assets, even if they are of the same type, or same configurations. This is apparent when creating similar building models such as homes. These model assets are referenced individually, and therefore, the memory should be significantly larger, even though the triangle, indices, and vertex count is much lower.

Table 27 shows the comparison between the procedurally generated house model and the user generated house model. The percentage comparison is also shown. The procedural model polygon count makes up 1.37% of the number of polygons within the user generated model mesh. The indices count and vertices count make up 1.37% and 3.19% respectively.

The user generated model mesh object has been obtained freely from a web platform; TurboSquid[66]. The procedural building structure is shown in Figure 106.

	Polygons	Indices	Vertices
Procedural Model	17	51	45
User Generated Model	1240	3720	1411
Procedural vs User Generated Model %	1.37%	1.37%	3.19%

Table 27 Procedural model and user generated model comparison of polygons, indices, vertices, and percentage comparison.

[65] http://xbox.create.msdn.com/en-US/education/catalog/sample/mesh_instancing
[66] http://www.turbosquid.com/

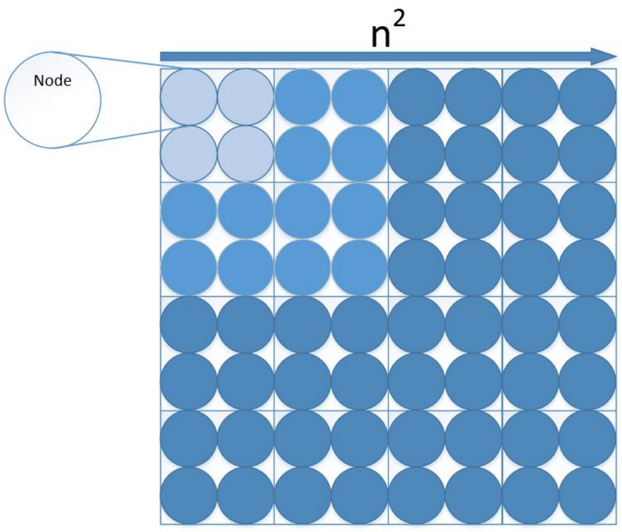


Figure 105 N squared representation of uniform scaling of nodes.

Figure 106 OSM map image of buildings of type 'house'. 641 house objects are within the extracted XML object. Image obtained from https://www.openstreetmap.org/search?query=trafford%20centre%20manchester%20uk#map=18/53.38993/-3.03696

9.4.1 Uniform Scaling Results

Within this section we discuss the results of the uniform scaling experiment. Table 28, Table 29, Table 30, Table 31, Table 32, and Table 33 show the results of the experiment. Within the experiment, issues arose which data is unable to be captured. This is shown with the result of NA. The reasons for this are due to Operating System restrictions with memory consumption.

Enough data is captured to extrapolate further results.

Table 28 shows the memory usage for each experiment. The amount of memory for empty nodes is minimal usage. The memory usage is recorded as a whole running program. This means additional assets are stored in memory, and instantiated classes. Before running the experiments, the memory usage is 328.2Mb.

As stated within section 4.1, the total number of tile within the OS mapping scheme equal 6,313,150. Generating PVSNodes which represent each tile will need large amount of memory. Using the results from Table 28 show that 1024 PVSNodes produce 419.4MB memory consumption. A single PVSNode produces 414.1MB memory consumption. This is a difference of 5.3MB. Dividing this by 1024 PVSNodes calculates 5.17578125 KB of memory is needed per empty node. Multiplying this value by the total number of tiles of the OS mapping scheme for the UK will produce;

6,313,150 tiles * 5.17578125KB = 32.675GB.

This calculation only contains the PVSNodes which represent the OS tiles. Adding the 93 PVSNodes for each 1km^2 which differentiate the OSM and terrain categorisations will produce;

6,250,000 1km^2 tiles * 93 categorisation PVSNodes = 860,250,000 plus the lower tile layers of 63,150. The total will equal 860,313,150. This value multiplied by the memory consumption for a single PVSNode will equal;

860,313,150 * 5.17578125KB = 4,452,792,670.8984375KB = 4.4529TB.

This is unrealistic that all PVSNodes would be loaded.

Section 4.1 also states the tile coverage of the UK will contain ~550,000 tiles. The memory needed for the empty nodes at this level will equal; 2.846GB. This is a more realistic memory consumption because as already stated, the OS mapping scheme includes a vast area of ocean which is not needed for our work.

The data attributes of a PVSNode is explained in chapters 6 and 7. The limit of the number of PVSNodes within regards to memory usage is still a concern when loading a country worth of data, but for small scenes of 10km^2 is reliable for even with a low hardware configuration. The memory usage for the procedural model mesh is encouraging. The duplication of 1024 nodes containing the duplicated procedural model mesh of a house model shown the memory usage is 1252.3 megabytes.

The user generated model mesh is considerably larger in terms of vertex count and indices count. The memory usage is considerably larger than that of the procedural model mesh. This is understandable due to the procedural model polygon count accounting for 1.37% of the user generated model, and 3.19% of vertex count.

The issue of out of memory exceptions is problematic for the 16^2 number of user generated models.

Table 28 and Figure 107 show a steady increase when n increases. At n = 1, the memory consumption is 427.5 for the user generated model.

The base memory consumption of the program runs at 298.7MB. Is has engine default assets loaded which is common for game engines and rendering engines.

The assumption of negating the base memory consumption of 298.7MB from the experiment base of n = 1 which is 427.7MB for the user generated model, the models memory consumption would have assumed to be 427.7MB – 298.7MB = 129MB. This is a wrong assumption due to scene set up attributes being populated. The model memory consumption can be inferred by dividing the difference between n=1 and n=2 which is 462.1 - 427.5 = 34.6 and divide this value by 2^2 which equals 34.6MB / 4 = 8.65MB.

Table 29 shows the percentage difference between increasing values of n with the user generated model mesh object.

Table 28 and Figure 107 show that when n = 16, the simulation runs out of memory when dealing with user generated models. The procedural generated model mesh does not run out of memory, even when n is equal to 32, resulting in 1024 separated model mesh objects. Having a scene which utilises both types of objects would improve memory performance. Careful consideration is needed as to how many user generated model mesh objects are to be loaded. A combination of many procedural model mesh objects and a few user generated models would be ideal. A LoD technique which swopped models for the user generated models which are closest to the virtual camera, or within direct eye line is needed. Models which are in the distance would be swopped for the procedural model mesh. This would improve realism, and improve memory consumption.

The FPS is a benchmark used for many real time applications. The FPS shows virtually no difference when dealing with nodes which do not contain a model mesh object. This is not surprising due to the aim of the evaluation is to investigate the changes in frame rates in relation to the use of model rendering. Table 30 and Figure 108 show the results of the FPS for the uniform scaling evaluation. If using only user procedural model mesh objects, the simulation can run at 27.3 FPS rendering, 1024 separate model mesh objects. This is an improvement than rendering the user generated model mesh object, which due to memory issues, the results could not be captured when the value of n = 16 or higher.

Within the user generated model evaluation, the FPS drops by 2.64% with each model added. This value is calculated by finding the percentage difference between the n = 1 and n = 2, which is 10.5812% and dividing this by the higher value of n^2 which is 2^2. 10.5812 / 4 = 2.64%. Calculating the same for when n = 2 to n = 4 states that the FPS for added model mesh object is 3.13. Additional overheads within the rendering pipeline may be the cause of the increase, or the architecture of buffering the vertex and index buffers.

Figure 108 shows the graph generated from Table 30. Using the graph, the maximum value of n which can be used to mimic the frame rates of the procedural generated model, when n = 32, is ~=

We have included the update of the scenegraph within this evaluation as shown in Table 31 for completeness. The results are the same for all maps due to members not being updated. Due to the optimiser for the material of the model mesh, eliminating updating of camera parameters if the virtual camera has not been modified from the previous frame, and combined with no need to update child nodes if parent nodes have not spatially modified, the update skip a majority of internal node updates; hence why the updating of a scene is vastly improved for many sized maps.

Table 32 and Figure 109 show the draw call in milliseconds. The table and graph show the increase in milliseconds of each draw call. Within the draw call the passing of the vertex and index buffers occurs. The passing of this data increase the draw call time when increased number of buffers are passed. Table 32 and Figure 109 show when rendering 32^2 procedurally generated model mesh objects takes 32.89 milliseconds. Using the user generated model mesh object, the draw call time increased by 87.5% when comparing the results of n = 8.

The GPU draw call shown irregular results. Table 33 and Figure 110 show that the GPU draw call for empty nodes to be irregular. The GPU draw call for the procedural model mesh shows an initial increase in call time, but then drops when n approached 8, and then increases as expected. The user generated model mesh starts at a high GPU call time, drops, and then rises from n = 4 to n = 8.

The irregular nature of the GPU call may be subject to OS background processes interfering. This was an initial concern, and all none essential processes where disabled during capture time.

We can conclude from this evaluation technique that the total number of procedural models which can be rendered at an interactive frame rate on a medium spec laptop is when n ~= 32. The use of roughly equals is justified due to the number of model mesh object can be increased, if only by a few more to keep the framerate to a reasonable 20 FPS and higher. As stated, the estimate of house models within a $1km^2$ area is 897.4. Using this estimate calculated, the area coverage when n = 32 is $1.14km^2$. (1024/897.4 = 1.14)

The recommended area coverage for the user generated model when n=8 equals $0.07km^2$ area coverage. The use of n=8 for this evaluation is chosen because results are not obtainable for n=16 or higher. Looking at the tables and graphs, the memory consumption, and FPS would still allow additional models to be loaded and rendered. Thusly n=8 for the recommended maximum size can be increased to a value between 8 and 16.

Both of these values are small compared to alternative applications. We state however the interactive nature of the members of the PVSNode, and model objects, along with the material objects attached to the model parts allow for greater interaction from a user point of view.

	1^2	2^2	4^2	8^2	16^2	32^2
Memory Usage Empty Node	414.1	409.14	415.9	419.07	419.8	419.4
Memory Usage Procedural Mesh	447.7	449.6	454.5	464.4	629.8	1252.3
Memory Usage User Generated Mesh	427.5	462.1	544	919.2	NA	NA

Table 28 Memory usage in megabytes for 3 types of objects; empty node, procedural model of a home, and a user generated model of a home. Object duplications are raised to the power of 2.

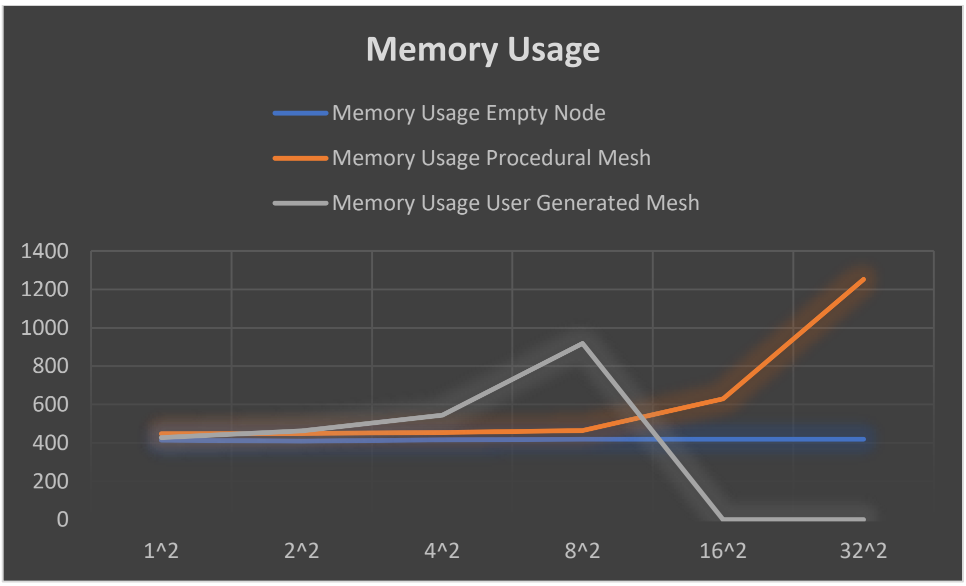


Figure 107 Memory usage graph.

	Values	Percentage
n = 1 and n = 2	427.5 to 462.1	8.09%
n = 2 and n = 4	462.1 to 544	17.72%
n = 4 and n = 8	544 to 919.2	68.97%
n = 8 and n = 16	919.2 to NA	NA
n = 16 and n = 32	NA to NA	NA

Table 29 Percentage difference between memory consumption and increase of n with user generated model mesh objects.

	1^2	2^2	4^2	8^2	16^2	32^2
FPS Empty Node	410.79	415.6	411.86	404.88	406.91	413.14
FPS Procedural Mesh	410.01	405.24	438.05	356.02	110.3	27.3
FPS User Generated Mesh	387.76	346.73	169.1	50.21	NA	NA

Table 30 Frames per Second for 3 types of objects; empty node, procedural model of a home, and a user generated model of a home. Object duplications are raised to the power of 2.

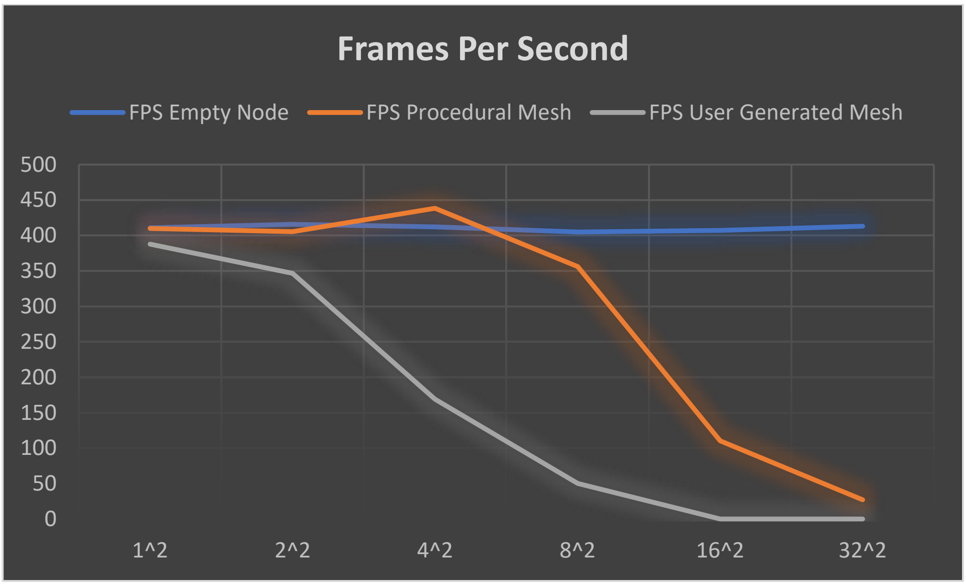


Figure 108 Graph for frames per second.

	1^2	2^2	4^2	8^2	16^2	32^2
Update MS Empty Node	0.01	0.01	0.01	0.01	0.01	0.01
Update MS Procedural Mesh	0.01	0.01	0.01	0.01	0.01	0.01
Update MS User Generated Mesh	0.01	0.01	0.01	0.01	0.01	0.01

Table 31 Update in milliseconds for 3 types of objects; empty node, procedural model of a home, and a user generated model of a home. Object duplications are raised to the power of 2.

	1^2	2^2	4^2	8^2	16^2	32^2
Draw Call MS Empty Node	0.31	0.24	0.33	0.24	0.24	0.34
Draw Call MS Procedural Mesh	0.44	0.89	1.28	2.35	8.8	32.89
Draw Call MS User Generated Mesh	0.59	1.41	4.55	18.8	NA	NA

Table 32 Draw call in milliseconds for 3 types of objects; empty node, procedural model of a home, and a user generated model of a home. Object duplications are raised to the power of 2.

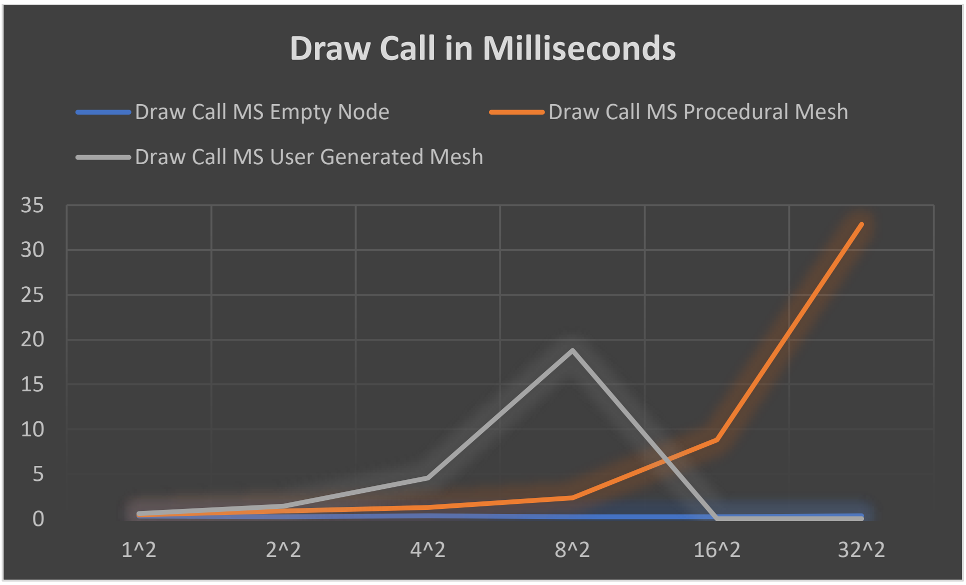


Figure 109 Draw call in milliseconds graph.

	1^2	2^2	4^2	8^2	16^2	32^2
GPU Call MS Empty Node	2.12	2.08	2.65	2.74	2.07	3.09
GPU Call MS Procedural Mesh	1.42	1.5	1.7	0.63	0.93	2.51
GPU Call MS User Generated Mesh	1.88	1.32	0.76	1.89	NA	NA

Table 33 GPU call in milliseconds for 3 types of objects; empty node, procedural model of a home, and a user generated model of a home. Object duplications are raised to the power of 2.

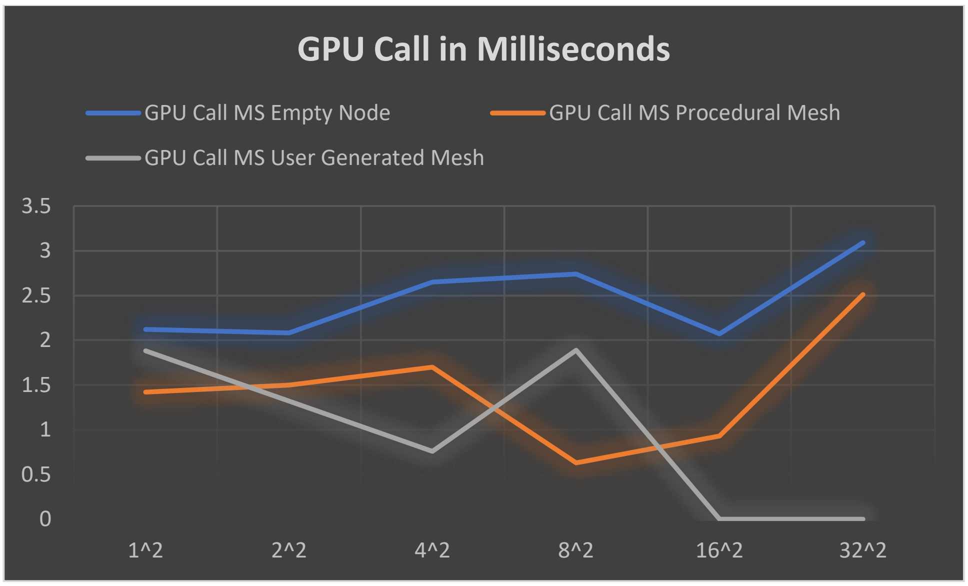


Figure 110 GPU call in milliseconds graph.

9.5 Scenegraph Evaluation

Within this section we will discuss the dispersal of data and data structures through the scenegraph structure. The data passed through the scenegraph will be a shader index value, and data structures used for light descriptions, materials, graphics state descriptions and effects.

The monitoring of the processing time taken to search for assets by user-defined queries will be undertaken.

Searching for an asset, a query is checking against nodes, and the type of PVSTag attached to the node. Searching a node, the properties of the node will be checked, i.e. all nodes of the scenegraph can be checked. If the user knows the type of the object (building, highway, amenity, and boundary) they can search by that type by selecting the OSMTag which has been applied to assets of that type. This allows the checking of whether a node has that OSMTag applied; if not, then checking does not have to commence.

Within the experiments which compare the 5 maps on the different platforms, the memory consumption would rise and fall dependent on which objects are being rendered. The terrain model mesh buffers holding large numbers of vertices, and having large polygonal meshes, the framerate drops when they are being rendered which is not a surprising fact. What is surprising is the memory consumption drops considerably. For this reason, we find it pertinent to record the rendering parameters of frames per second, update call in milliseconds, draw call in millisecond, and GPU call in milliseconds, and memory consumption, for rendering everything within a scene, and everything but the terrain.

9.5.1 Search Cases upon the Scenegraph

The scenegraph is a valid choice when working with geographical nodes and nodes of spatial parameters. The combination of allowing users to search for objects within the scene, from node or model mesh, or a node which has an OSMTag attribute pertinent to a particular type (Building, Highway, Amenity, Boundary), by single or multiple parameters is problematic: depending on the size of the dataset.

The extraction time for map 3 which contains 1427 nodes takes: 0.0003603 milliseconds. We can extrapolate this result to estimate the time for larger Scenegraph structures.

Within this section we will discuss the search cases which the scenegraph is suited for, or where a various type of data structure may be an optimised choice.

The fire brigade may wish to search for buildings which are over 50 meters tall and fireproof. The police may wish to visualise highway structures which are at least 3 lanes wide as to allow oversized vehicles through.

Table 34 shows the processing time of functions used upon the scenegraph structure. The nodes traversed is also counted and referenced. Map3 is used due to maps 4 and 5 throwing out of memory exceptions. The time recorded shows that searching relatively large scenes allow for real-time searching.

Classifying objects by type allows the iteration of extracted nodes. The extraction and sort function takes 0.0005657 milliseconds for this map size to complete.

The searching for the Building is by name “Royal Liver Building”. The searching for high way will be an unclassified type with name of 849. This will also be searching by PVSNode only.

Function	Processing time in Second
Disperse shader index to all of tree	0.0007791
Disperse light description to all of tree	0.0008727
Disperse graphics render state to all of tree	0.0005148
Disperse a material controller to all of tree	0.0221018
Disperse a spatial controller to all of tree	0.0006565
Search and return assets from query on referenced nodes	0.29225
Search and return buildings from query on all of tree on referenced nodes	0.14099
Search and return highways from query on all of tree on referenced nodes	0.24391

Table 34 Processing time of scenegraph functions for real-time look up.

The algorithm for the dispersal of the shader index value is shown in Figure 111. The input for the algorithm is the integer shader index value and the node for which the integer value will traverse from through the scenegraph, as well as the model mesh hierarchy to the PVSMaterial objects of each model part. The node acts as the base of the scenegraph structure dispersing the data down through the tree, allowing any node to act as a category which can have the shader index value set. Each child within the node will have its internal setShaderIndex function called. The output of the algorithm is all sub nodes of the scenegraph have their shaderIndex values set on the material of any model part. Allowing the changing of a scenes rendering within a single function call.

```
Input
int shaderIndex
node scenegraphNode

Start
        if scenegraphNode has a model object
                foreach model part in model
                        set shaderIndex on material
        foreach child in scenegraphNode
                set shaderIndex

End

Output
shaderIndex set on all nodes model mesh part material objects, changing rendering of
scene
```

Figure 111 Pseudo recursive algorithm for dispersal of shader index upon scenegraph structure.

The algorithm for the dispersal light description is shown in Figure 112. The input for the algorithm is a PVSLightDescription which hold the information pertinent for a virtual light. Along with this is the index to input the lightDescription data structure into the internal array of PVSLightDescription's. The node is the base of the scenegraph for which the algorithm will recursively disperse the PVSLightDescription through the sub nodes of the scenegraph. A Boolean is used to check if the lightDescription should be cloned. Cloning the object will create a new instance of the light description, meaning each light clone will be independent of each other. If the lightDescription is not cloned, then it will be passed by reference meaning updating of a single lightDescription, through the IVI for instance will update the light parameters on all model parts which reference the lightDescription object.

```
Input
PVSLightDescription lightDescription
int lightIndex (0 to 2)
node scenegraphNode
bool clone

Start
    if scenegraphNode has a model object
        foreach model part in model
            if clone is True
                copy lightDescription to new object
                set cloned object to material of model part at lightIndex
            else
                set lightDescription to material of model part at lightIndex
    foreach child node in scenegraphNode
        set light description on children
End

Output
lightDescription changed at lightIndex on each nodes model mesh parts material
```

Figure 112 Pseudo algorithm for dispersal of a PVSLightDescrition structure through the scenegraph.

The algorithms for the dispersal of the graphics render state, material controller, and spatial controller are the same as the algorithm to disperse the light description struct but removes the setting at a specific index of an array. This is because the data passes is stored as a single object, or stored in a List<T> structure and thusly, no indexing is needed.

9.5.2 User Case Queries and Scene Features

Within this section we will discuss and present the evaluations for common uses for domain experts; fire-brigade, police, ambulance service. We present initial cases of realistic searches upon the scenes.

Firstly, we highlight some screen captures from the simulation. Figure 113 highlights the use of shared lighting on model assets. Top left shows all assets rendered using a basic shading effect with the light a predefined position. Top right shows the same assets but rendered slightly differently due to the change of direction from the scene light. Bottom left shows a scene without the DSM terrain assets. Bottom right shows the DTM, and procedural assets rendered with textures.

Figure 114 shows the comparison between user generated models and the terrain DSM and DTM models. Top left shows only the terrain of DSM and DTM. Top right shows the combined user generated model of the Royal Liver Building. Notice the positional differences. The differences can be used to modify the original user generated model mesh. The middle right shows the textured model textured terrain. The bottom image shows a higher view point and the Royal Liver Building at a large scale to showcase the ease at bringing attention to a single asset.

The police may search for all highways which lane count is equal or over the value of 3. Within map 3, the search produced 24 highways which satisfy this query. The search took 0.0646034 seconds. The dispersal of a material controller and spatial controller took 0.0000949, 0.0000222 respectively. The highway highlighted is shown in Figure 115.

The fire-brigade may wish to search and highlight all buildings which allow a capacity of 100 or more, and is over 50 meters tall.

The fire-brigade may wish to search and highlight all buildings which are over 50meters tall and are not fireproof. Within map 3, 5 buildings are found using this query, and took 0.00745775 seconds to find through iteration of extracted nodes. The dispersal of a material controller took 0.00009771 seconds to complete, and dispersal of a spatial controller took 0.0005738. The buildings found are shown in Figure 117.

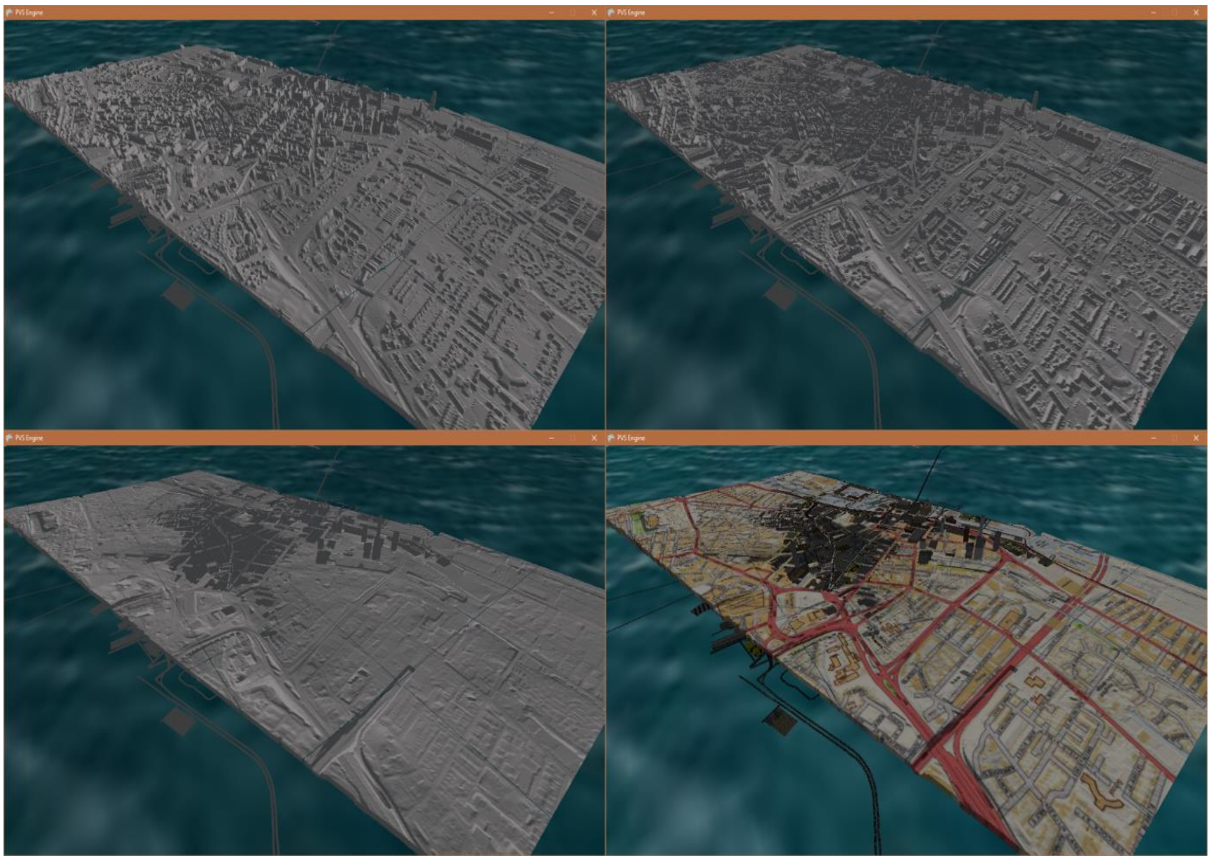


Figure 113 PVS simulation. Top left is basic rendering. Top right shows the difference when a light position is modified. Bottom left has DSM terrain removed. Bottom right has terrain and assets textured.

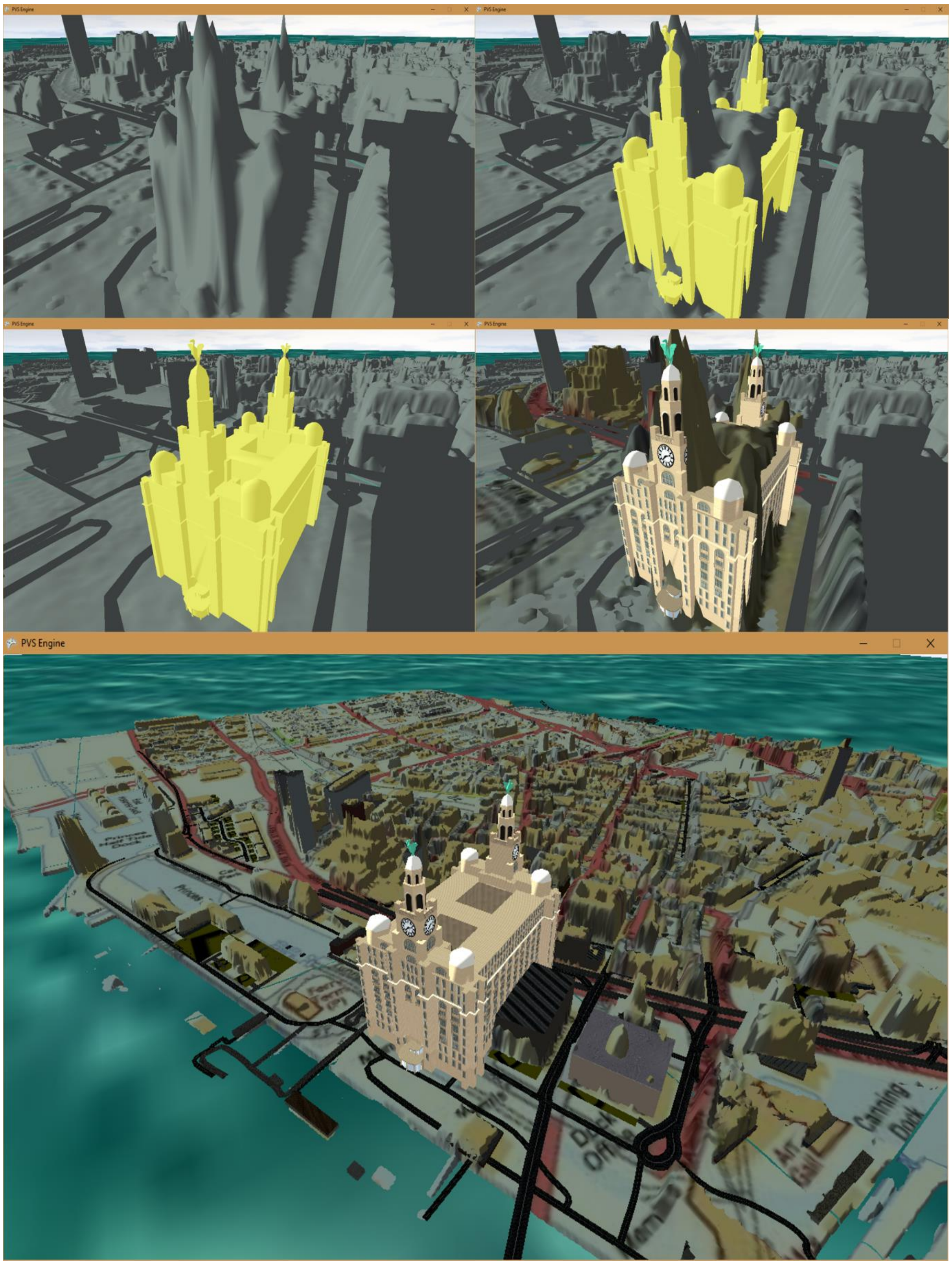


Figure 114 Screen capture of simulation.

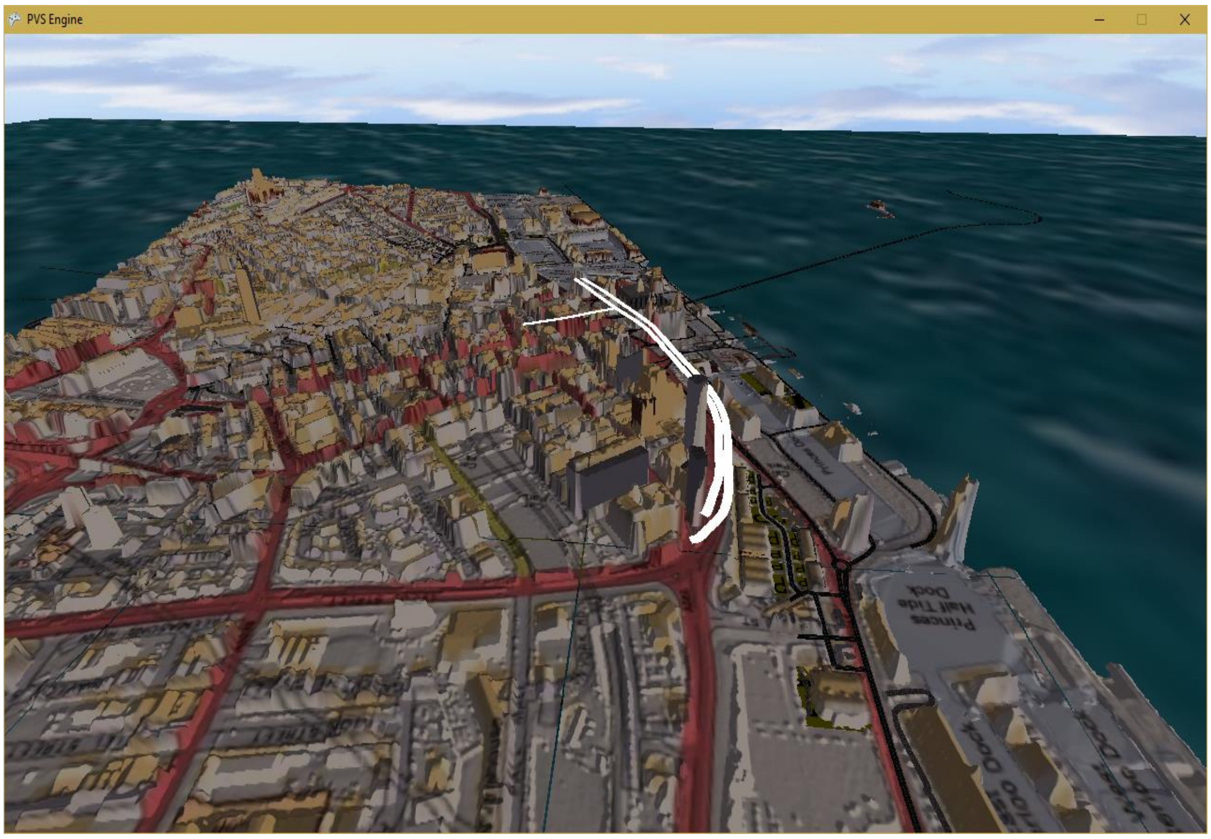


Figure 115 Highlighted highway structure from user generated query.

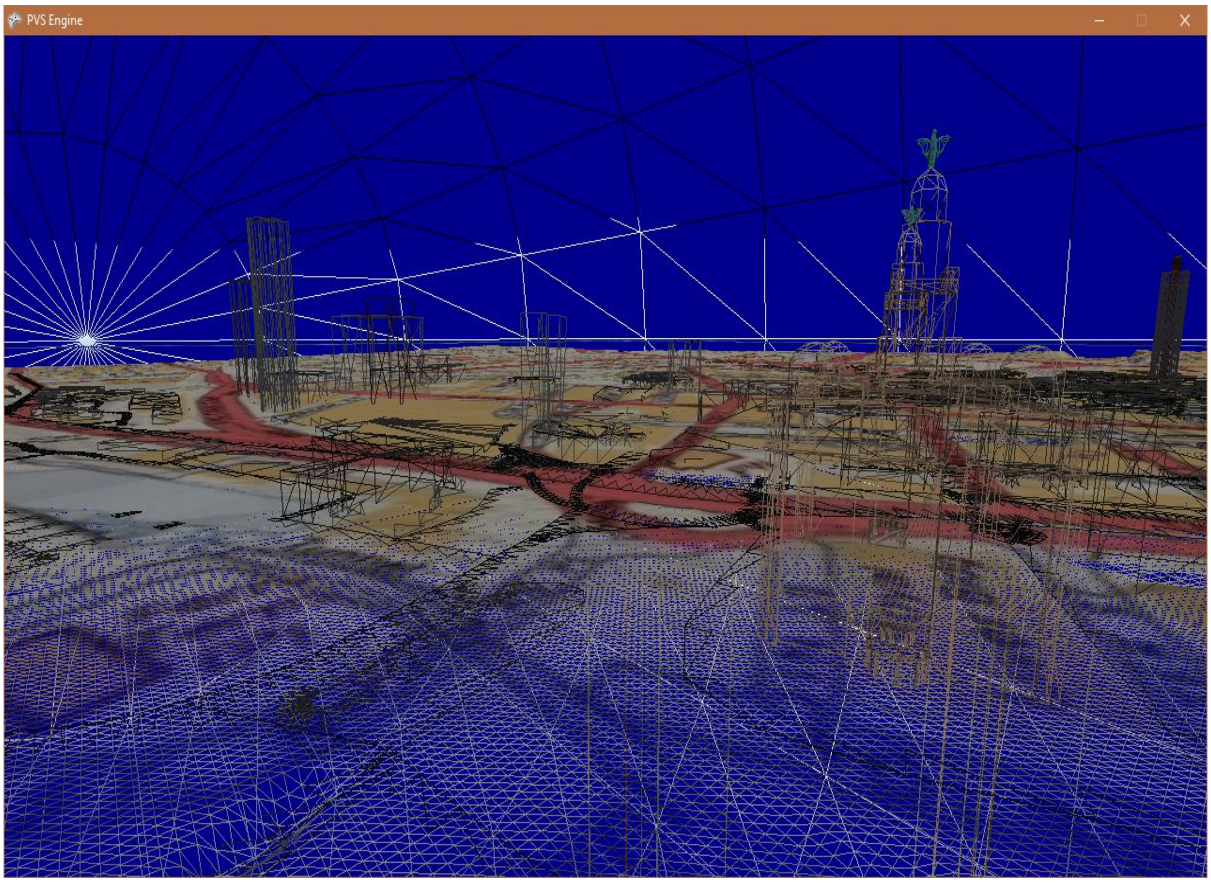


Figure 116 Wire frame scene.

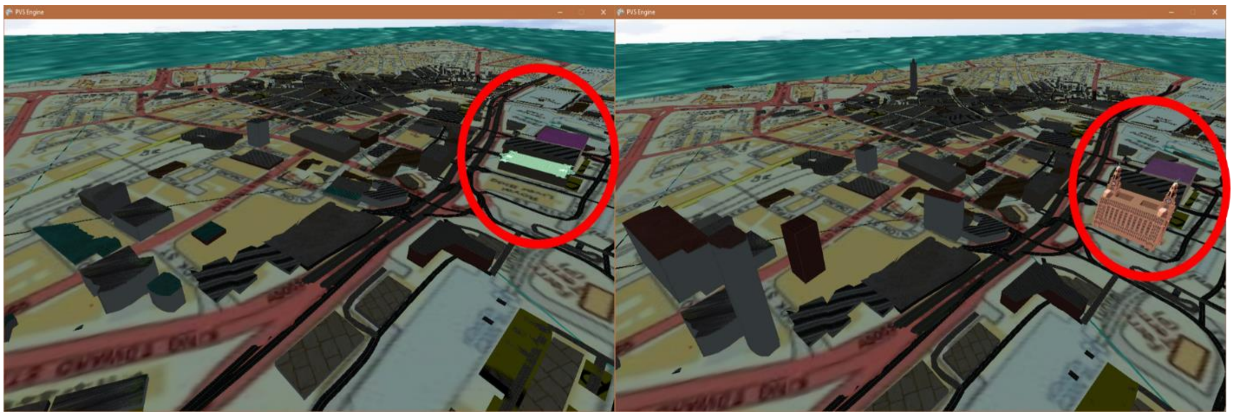


Figure 117 Material and spatial controller applied to fire brigade query.

9.6 User Interface Evaluation

Due to the integrity the IVI brings the application and interactivity to the framework and the algorithms we state that the IVI is determinately needed. A user can interact with scenes, changing properties of individual models, as well as change the rendering techniques and render states applied to each of the models.

Multiple IVI screens have been developed to gain access to the assets within the scene, and gain access to the data structures used to organise, structure, and visualise the scene; novel scenegraph, novel designed IASF, global lighting manager, and materials of models.

There are a number of forms which act as main elements of the IVI; main form, node form, and model form.

The main form contains tabs which allow access to the scenegraph visualised as a listed tree view. A tab allows access to the global light manager and its 3 lights. These lights can be modified in both spatial properties and descriptive properties. The lights can be dispersed to the model part materials for use within rendering the model, through the scenegraph. Another allows access to the camera system used for modifying the projection parameters of the camera within the scene as well as extracting information from the camera used for debugging. The main form contains a search tab which allows users to generate queries and search the scenegraph. This tab provides the generation of spatial and material controllers to be applied to the searched for nodes. Within the main form is a location tracker.

A node is selected from the Main form tree representation of the scenegraph which displays a form containing the information of the node. The nodes spatial properties can be modified, and prebuilt controllers can be applied to it. User generated spatial controllers can also be applied to the node. Material controllers and GPU render stated can be created and applied to all model parts of the attached model. The model information is displayed in a tab within the form. This allows access to individual model parts which in turn displays a form for the Material object.

Within the material form, a user can modify all properties which are applied on the GPU; ambient colour, diffuse colour, intensities, and shader index and others. This allows the display of GPU parameter data used for testing. It also allows individual material controllers to be applied to the model part.

The reliance of the .Net framework restricts the development of the IVI to only .Net framework availability. This dependency should be removed in future iterations of the framework. A custom IVI library should be created. This will improve scalability and flexibility, as well as code reuse for alternative applications and APIs. The use of WinForms application is still a valid option for proof of concept and rapid prototyping which was needed for this project.

9.7 Comparison

Within this section we state the comparisons between the PVS framework to that of Google Maps, ArcGIS, OSM, SAVE, the ALLADIN project, and the results of OSM2World. The comparison of features, components, scalability, flexibility, realism of visualisations, and data representation will be shown where applicable. If a competitor does not contain a feature we will state that we have no comparable evidence. If the PVS does not contain a feature, or component found in a competitor, we will issue what needs to be done to acquire such a feature, or if the feature would not be needed.

We will keep this section brief and give bullet points with small explanations into the comparisons between PVS and competitors. If we have already stated the benefit of our system over a competitor's component or feature, we will not repeat ourselves for the other applications.

9.7.1 Google Maps

- Google maps limit the interaction with the camera placement and navigation, PVS does not.
- Google maps allow for satellite 2D top-down images and OS style 2D top-down images, as well as zooming of tiles. PVS technically can have this if the camera is positioned above the scene pointing down and putting the camera into orthographic projection state. PVS system has more dynamic camera system which allows cameras to be translated, and modified to how a user wants, viewing scenes in a large number of angles.
- Google uses LoD techniques to remove objects from a scene. If PVS is to view as large a scene as Google maps then it must implement LoD techniques to reduce the terrain vertex buffers to not overwhelm the hardware or software.
- Traffic views for major roads are available for Google maps. In PVS, only coloured lines are available at four settings from *fast* to *slow*. This data can be incorporated with additional coding through OS traffic data. If this data is incorporated into the PVS framework, single lines depicting the speed of traffic at varying times can be shown. With PVS, the highway model meshes can be animated with pulsating colours, highlighting real-time changes in data. Google Maps currently do not have this ability. Only data and a parser function are needed for PVS.
- 2D buildings have a shadow effect added which sometimes does not line up with the 2D building representation. Additional shadow rendering techniques can be added to the PVS framework with the addition of Deferred Rendering. Only a vertex and pixel shader function is needed to be added to the IASF, as well as creating a render target. The setting of the graphics card state to record depth can already be set.
- 3D view of scenes can only be projected using the satellite images. We state that this would not be sufficient enough to be classed as 3D, but 2.5D projection. The camera can only be

set to 90° intervals. The PVS framework allows 3D viewing at any angle or direction the user wishes, as well as in any projection; orthographic or perspective at variable field of view.

- PVS does not currently include satellite imagery. When using Googles Street View, the user can jump to locations, and the projections of images captured from moving vehicles are used. PVS allows street view, but by use of a fully movable camera system, and 3D generated scene. Googles street view may provide accurate images, but PVS provides further flexibility with viewing a real-world scene from a high level perspective, manipulating assets in real time.
- Both systems lack the complete global coverage of data; streets, buildings, amenities, images, geo-data, and many others. Google does not contain all street view images, but it does contain a large portion of satellite images, whereas OSM contains much user generated content; thusly the PVS simulation output will contain this user generated content and therefore present much more informal data which is relevant to users.
- Google maps provide a terrain view with the retraction of buildings and other information not projected using satellite imagery. The image on screen is modified to add shading corresponding to heights of terrain. The variance is most noticeable when zoomed out to a high level. This is limited. PVS with the use of OS or LiDAR data provides improved terrain visualisations up to 25cm points; if interpolated as the UK government does with its OS terrain maps.
- Google maps allow the addition of user generated photos. This function extends the information available to the user, giving insight into the different views of a building or a structure. This functionality is not available in PVS. The framework does allow the addition of textures to model asset parts.

Google maps is a main competitor to this work. In conclusion the PVS framework allows for improved interactions with assets generated from GIS data sources, and allows data overlay in the form of animations and visualisations, where Google maps do not.

9.7.2 ArcGIS

The ArcGIS slogan on their webpage is *'Mapping Without Limits'*. Their framework works as a cloud based platform. Being cloud based, their simulations are scalable and capable of real-time calculations due to increased hardware and software availability. The PVS is scalable in the confines of the UK, but is adaptable with additional development.

They also provide many individual apps which can run within browsers, desktops, and mobile devices. The PVS framework is developed for desktop only. The PVS framework does not work on mobile devices or web platforms without conversion from XNA to MonoGame[67].

ArcGIS provides many base maps; Shaded Relief with Labels, OSM, Imagery Hybrid, National Geographical Map. All these base maps have 1 thing in common; they are all 2D top-down map representations. PVS advances this by creating the 3D environment needed for visualisations.

ArcGIS allows open access too many of their libraries for further custom development by users. The ArcGIS is a platform which allows users to used premade maps, or allows the definition of their own maps. Prior knowledge is needed to configure the ArcGIS to generated maps needed. Our framework allows the customisation of maps at runtime, for a pre-made scenes. PVS is designed to mimic real world interactions with objects and employ game controller inputs which allows users to quickly become familiar with the interactions of the simulation. Our approach allows greater flexibility, with minimal of domain knowledge; unlike that of ArcGIS.

9.7.3 Open Street Map

The framework and simulation is built utilising OSM data as its main data-source for which scenes are built. The web portal allows the viewing of data in a top-down 2D perspective, same as Google Map, with additional layers of information and additional maps specifics; cycle maps, highway maps, and the ability to highlight selected boundaries to bring up additional information. Multiple maps need to be viewed as separate maps and cannot be hierarchically layered together to view on top of one another. As stated, the PVS framework allows the viewing of scenes but not as a 2D perspective, unless the camera system is specifically parametrised and set as orthographic, and can view multiple user selected data sets within a single scene visualisation.

PVS can highlight varying assets within a scene to create layers within the visualisation to mimic the multiple map layers of OSM such as cycle paths. This is done by querying the assets within the scene by user defined parameters. A user can select cycle paths of OSM, or search all Highway type roads which allows bicycles and highlight them green. An added benefit of the PVS framework is the multi highlighting of selected assets. If a user wanted to add a theoretical layer, they would just search for the assets they want, and change the colour of said assets. This would create a multi-layered visualisation. This combines the multiple maps provided by OSM, but within one single visualisation.

[67] http://www.MonoGame.net/

9.7.3.1 3D OSM

There are many 3D platforms building from the 2D and 3D geospatial data within the OSM database. These platforms focus mainly on the generation of buildings. Some are used within desktop applications (OSM2World[68], OSM-3D[69]), while others are through web-browsers (OSMBuildings[70]). We compare the features and renderings of our framework with that of OSM2World, OSM-3D, and OSMBuildings. The applications vary by developer and features contained, as well as the age of development.

The PVS framework is superior to the applications stated by the allowance of individual assets modifications and shader augmentation. The applications allow users to view scenes as a whole but limit the users' interactions with a scene and assets.

OSM2World does utilise Ray tracing (POVRay[71]) rendering techniques whereas the PVS framework does not. This feature does allow reflections of scenes within water and windows. The PVS framework cannot do this as yet due to its forward rendering. The addition of deferred rendering and additional vertex and pixel shader functions added to the IASF is capable. This is a feature which will improve the visualisations of real-world virtual scenes. The scenes of OSM2World can be exported and imported in external 3D modelling assets; Blender. The current implementation of PVS does not integrate highway markings which a small number of highways within the OSM database contains, whereas the OSM2World generates highway 3D models with markings which direct traffic. The PVS framework will need further development to include markings on the highway, but the 3D highway models are generated by querying data within the OSM database. The visualisations of OSM2World are limited but do allow a 'debug' view which views the wireframe models of the terrain and assets within the scene. The PVS framework provides many visualisation permutations through the use of the IASF, the material parameter animations, the spatial parameter animations, and the ability to change GPU device state (wireframe, transparent, opaque, etc.).

9.7.4 SAVE

The visualisations of the SAVE project are considered our main competitor in respect to user generated visualisations. The SAVE project allows users to create visualisations of particular needs and wants depending on the user's individual wants and needs which are assign to Eigen vectors as percentages. The scenes of SAVE are created through manual model mesh creation, and the individual model parts of buildings material shader parameters are assigned the values created by

[68] http://osm2world.org/
[69] http://www.osm-3d.org/home.en.htm
[70] https://osmbuildings.org/
[71] http://www.povray.org/

the Eigen vectors and parsed to shader parameters inputs. We state the PVS framework is capable of the same but through a varied pipeline. The user can combine textures (diffuse, normal, specular), and other shader parameters and apply them to models. We have extended the techniques of the SAVE project to include animations of varying styles to create a large number of effect animation permutations.

PVS can create scenes of any location within the UK if data permits. The SAVE project concentrated on creating a model mesh scene of one location and work solely on a single building. The PVS can apply the effects onto any model mesh asset within a scene; terrain, buildings, highways, etc. If the model mesh data is available from the SAVE project, the same scene can technically be generated and compared further. This is not the case.

9.7.5 ALLADIN

The comparison between the ALLADIN project and the PVS framework is slim. We evaluate that the combination of both systems within a future project will benefit both ALLADIN style projects and real-world visualisation projects as PVS.

The ALLADIN project allows users to view agents working within the world space of real-world highway structures only in 2D top-down. The PVS framework can and should be extended to include artificial agents and view them simulating within real-world virtual environments.

9.7.6 OSM2World

To generate elevation terrain models, OSM2World utilises the object attributes of OSM, such as the ‘incline’, ‘layer’, ‘tunnel’, ‘bridge’, as well as the absolute elevation attribute held within OSM.

Generating a 3D model with OSM2World takes some knowledge of computer systems due to its use of the command prompt and lack of IVI. We state the process to create an .OBJ model mesh structure using OSM2World using a Windows Operating System. To create a 3D model mesh, the documentation states that the Java Runtime Environment (JRE) be installed and to unzip and merge multiple downloads from the OSM2World website. The user must use the command prompt to invoke the building from OSM xml file to an .OBJ model mesh file. Multiple command prompt configuration invocations can be called to vary the output.

We will generate and compare 5 maps; each map increases is dimensions. The maps are of Liverpool UK and contain a mixture of characteristics; contains water, gradual incline in sections but a large drop from land to the river, and contains a mix between primitive buildings and complex building structures, and a multitude of highway structures.

Table 35 shows the statistics for the model mesh object exported from OSM2World through the command line. All elements are pooled into a single vertex buffer for rendering.

Figure 119 shows the processing OSM files by OSM2World visualised within 3DS-Max. OSM2World does not include a model viewer and thusly 3DS-Max is used to view the scene. 3DS-Max shows the information within the model; polygons, triangles, edges, vertices, and we have extracted the FPS. This information for each of the maps is shown in Table 35. OSM2World pools together the vertices of all objects within a scene into a single model mesh object. This restricts the use of and identification of individual assets; a benefit which is included with the PVS framework.

We have not included the time it takes for processing the OSM maps using OSM2World due to the fact that the speed is significantly faster than that of PVS. We are also unable to monitor the speed of the processor apart from monitoring the time is takes from starting the process, and waiting until a file is outputted within the directory. The time we monitored for the processing of the largest OSM map was roughly 30 seconds. We iterate as well the use of Visual Studio will reduce processing speeds of our pipeline, and the additional checks that our work makes against the data within OSM.

OSM2World does not check and extract information from OSM to the degree which we do. We also use random selection techniques as to make sure models within an environment do not share the same texture or material parameters as other models within a scene. The noise added to the default data adds an element of realism to a scene. We iterate, as stated within our background research that the human eye is adapted to seeing repeated patterns and elements within a scene. OSM2World uses the same colour values for the sides of the buildings for all buildings, disregarding their type.

Z-fighting is an issue with polygons which overlap each other, especially highway model mesh objects upon a flat terrain. Within PVS, noise generation techniques are used for our models to remove, or limit the issues of z-fighting. The addition of random decimal values less than 0.001, are added to polygons ahead during processing, as to limit polygons lying exactly on top of each other. OSM2World places polygons within the same parameters, and thus popping between the polygons occurs; especially within a dense area of individual buildings such as shopping centres.

Figure 119 shows the visualised OSM maps within 3DS-Max. Figure 118 shows the issues we wish to highlight in the model mesh object generated by OSM2World.

Within the figures we can see issues and alternatives from PVS.

- Building's rooftops coloured red, and are flat. Within our results, the rooftops have a random element to them, varying the height position of the vertices the rooftop is made up of. This adds an element of variation to the visualisation and environment. We also add textures applied to the rooftops. Each rooftop is a model part, and thusly can be modified, and altered, or have a controller applied to the rooftop to highlight the building.

- The railway highways are placed on the same height plane as all other model mesh objects. Within our visualisations, we check the OSM level attached to the highways. If the level is 0, it is placed on the terrain. If the level is 0 to -5 then it is placed below the terrain. 0 to 5 is above ground.
- Within Figure 118, the trees shown are made of primitive models. Handmade model mesh objects used in place of the procedural generated primitive models.
- The top middle image shows a large gap within the terrain not present. This is not shown within other generated models shown in Figure 119.
- The top right image shows a floating building part. This is an issue, and it not this single instance. We noted 3 floating building parts. Allowing custom model assets to be used will alleviate this issue.
- Within the bottom left and bottom middle images, the highway structure issues are shown. The highway polygons are set at the same Y value, and thusly z-fighting occurs.
- The issue shown within the bottom right image, the model mesh is shown to be using per-vertex lighting. As OBJ files do not contain information pertinent to stating whether per-vertex or per-pixel rendering should be utilised, we assume that 3DS Max is using per-vertex shading. This is an issue. Within our framework, the algorithms try to render objects using per-pixel shading. If this is too intensive, then rendering techniques all are altered accordingly. This provides the initial high resolution of rendering, before opting for the lower realism of per-vertex shading.

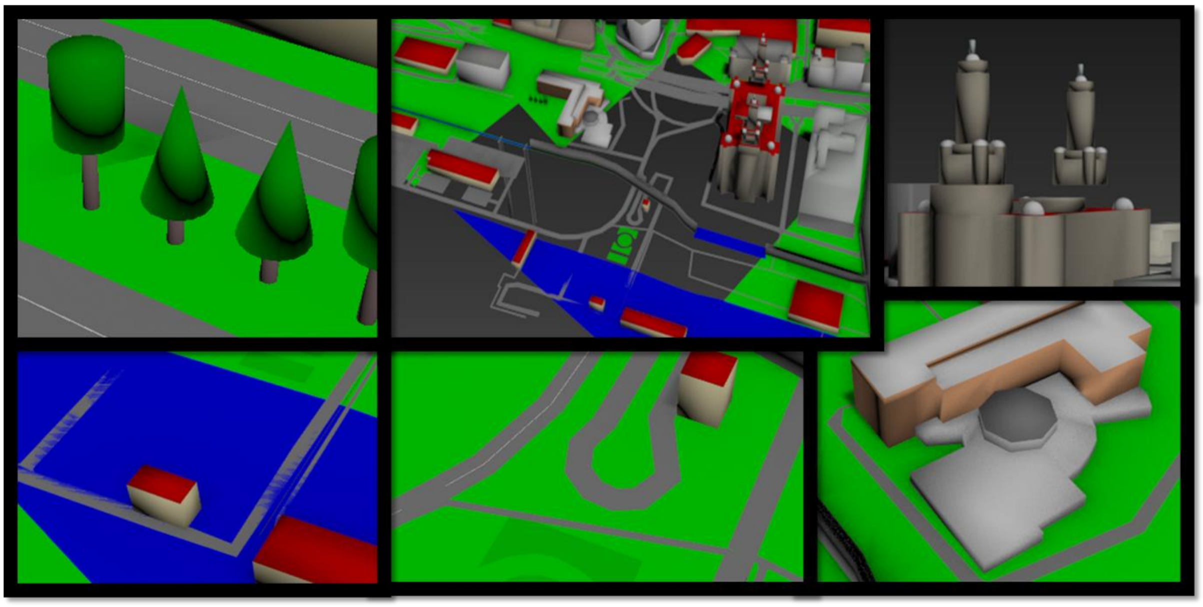


Figure 118 Issues within the OSM2World visualisation. Images obtained from OSM2World.exe application results.

Map	Polys	Verts	FPS
01	20,980	11,950	57.587
02	49,766	25,793	45.694
03	97,815	56,851	33.244
04	178,188	101,587	29.298
05	260,049	144,641	16.195

Table 35 OSM2World and properties.

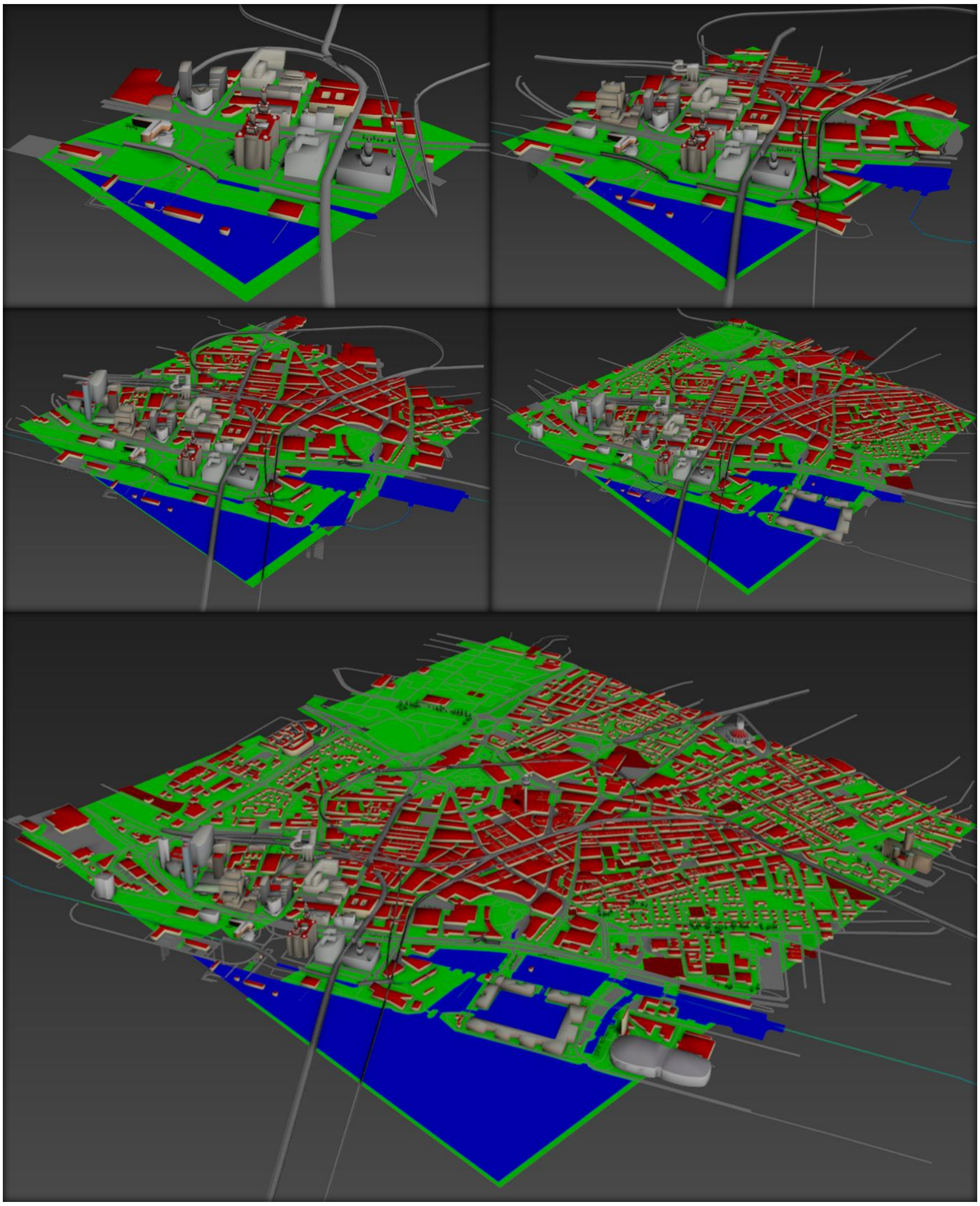


Figure 119 OSM2World models within 3DSMax. From top left, to bottom the maps increase in area coverage.

9.8 Feature Comparison

Within this section we will discuss the features which are provided by the alternatives, and within our framework. Table 36 shows the comparisons between PVS and similar projects, platforms, or frameworks.

We reiterate the features of a GIS stated within section 2.3;

- 2D and 3D Camera system with multiple viewing projections, and controllable by user
- Using satellite imagery, and other 2D/3D content to bring realism to scenes
- LoD to improve large scene rendering.
- Real-time framerates
- Scalable development tools/API
- Allow importing of user generated content
- Realistic procedural assets
- Advanced rendering techniques applied to scene and individual objects
- Object selection and modification
- Geospatial data visualisation

We state the division of the features into sub features. For example, 'traffic views' can be categorised under geospatial data visualisation. Scalable development and scalable implementation is categorised as an open API which the framework is. We have entered the feature of 'Browser Support' due to a majority of providers have browser support.

Green boxes are for features which the project has. The yellow is where there is ambiguity into the feature, e.g. the PVS framework can have satellite imagery within the visualisations, if they are available. Red is for features which the platform does not have. We have included the totals. The most features available is within the PVS framework, with ArcGIS have 1.5 features less, and ALLADIN having the least. The OSM2World framework provides the least amount of features. Figure 34 shows the feature count.

We have stated that Google have ambiguity with a 3D camera, with perspective and orthographic projections. This is due to the maps are projected which the camera being angled, this does not allow full 3D viewing of buildings. The SAVE project resembles our visualisations and animations applied to mode parts through shader manipulations. Table 36 shows that the SAVE project lacks many features which are available within the PVS framework. The increased number of features available to users allows the flexibility to view virtual scenes in an increased number of perspectives and visualisations; further than the SAVE project, and OSM2World, and the others.

Features	PVS	Google	ArcGIS	3D OSM	SAVE	ALLADIN	OSM 2World
2D	0.5	1	1	0	0	1	0
3D	1	0.5	1	1	1	0	1
Perspective Camera	1	0.5	1	1	1	0	0
Orthographic Camera	1	0.5	0.5	0	0	0	0
Satellite Imagery	0.5	1	1	0	0	1	0
Traffic Views	0.5	1	0.5	0	0	0	0
Scene LoD	1	1	0.5	0	0	0	0
Model LoD	0.5	0	0	0	0	0	0
Realistic Buildings	1	0.5	1	0.5	1	0	1
User Navigation	1	0.5	0.5	1	1	0.5	0
Scalable Development	1	0	1	0	0	0	0.5
Scalable Implementation	1	1	1	1	0	0	1
User Generated Content	1	0.5	1	0.5	0	0	0.5
Browser Support	0	1	1	1	0	0	0
Interpolated Highways	1	0	0.5	0	0	0	0
PGC	1	1	0.5	1	0.5	0	1
Advanced Rendering	1	0	1	0	1	0	0
User Applied Animations	1	0	0.5	0	0.5	0	0
Real-Time User Modification	1	0	1	0	1	0	0
Visualisation	1	1	1	0	1	1	1
Total	17	11	15.5	7	8	3.5	6

Table 36 Feature comparison between PVS and competitors.

9.9 Evaluation Conclusion

Within this section we have stated the comparisons between the competitors and the PVS framework, and presented as well as discussed the evaluation techniques/results of the PVS framework.

We state that OSM is a valid option to use for the creation of real-world urban virtual environments. The data set is suitable if the data is thoroughly checked and altered. Alternatives to our framework which give similar results are OSM2World and the SAVE project as well as the commercial GIS of Google maps. OSM2Worlds advantage over the PVS simulations is its speed of processing. While our processing takes longer, the results allow greater flexibility with scene rendering, and user interaction. Assets within the world are separate entities, and can be modified, as well as information extracted and shown to the user. OSM2World outputs a single pooled model mesh object which allows a high-level overview to real-world scenes. Our work goes a further step into the LoD into a scene. The LoD may not be as detailed as the work of Bo Moa[11], [25] in its current state, but the addition of further user generated content will alleviate this issue. Bo Moa work does not allow for user interaction within the scene as does PVS and that of the SAVE project. The PVS framework allows further interaction than that of the SAVE, and allows encoding of user weighted data and values by the use of spatial and material controllers which can be applied to any and all model objects within a scene; a scene which can generate any location within the UK.

The uniform scaling evaluation gave initial values for the number of empty nodes, procedurally generated model mesh objects, and user generated models, which can be rendered within a scene. The results determined that a mix of procedurally generated assets and user generated content can provide realistic scenes of over 1km^2 with terrain models. This does not seem large compared to OSM2World, but the increased overhead of storing individual small model mesh buffers, and PVSNodes restricts the number of assets to be rendered to screen. The pooling of vertex buffers for objects which store the same data is needed to increase framerates, and reduce rendering overheads. Hardware instancing should also be included to render the same model multiple times to reduce overheads.

The use of the IASF improves the rendering capabilities as stated in section 5.3.1. The structure layout allows additional shader functions to be added to the effect shader file, and small changes to the code base of the framework. It improves rendering rates, and allows dynamic reconfiguration depending on real-time monitoring parameters, such as frames-per-second.

We conclude that the algorithms in place allow visualisations of real-world urban environments in real time and allow non domain users to quickly interact and visualise queried assets within virtual scenes.

10 Conclusion and Future work

Within this work, we have under gone much research into different domains of GIS, and Computer Games Technology for the combination and generation of two systems which when together, creates a framework to combine erroneous GIS data for the generation of inferred procedurally generated 3D virtual scenes.

From the objectives of the project, we state that all aims of the project have been either partially met or fully met.

Aim 1, "*to review and consolidate the knowledge and state of the art on big geospatial data rendering and visualisation*". This objective has been met within the background and literature research carried out through the project. The Agile methodology chosen, benefitted the project by re-planning, and researching into issues found during the development process.

Aim 2, "*to identify and collect the real world data to generate complex real world 3D environment*". This aim has been satisfied by the use of OSM data providing a large amount of detail and relational geospatial as a base for virtual scene generation. The additional of LiDAR data improves scene visualisation by the addition of rigid terrain data in the form of DTM and DSM. LiDAR being erroneous needed additional data as a base to combine and override the erroneous or missing LiDAR data. Combining complete terrain data with OSM boundary data, sub data sets representing buildings, and locations can be extracted and converted to model mesh representations of the primitive OSM data representations.

Aim 3, "*to design framework and software components to develop a big geospatial data rendering and visualisation system*". This aim is satisfied by the generation of both the pre-processing system, and the runtime simulation system.

Aim 4, "*to create novel data structure to combine several data sources and design new algorithms to process and visualise these data using a 3D game engine*". This aim has been satisfied by the creation of multiple algorithms to process the big geospatial data sets into complete data sets for the combination, extraction, and inference of additional data for the creation of 3D model assets to visualise and render real-world virtual worlds. To generate model assets, procedural generation techniques have been utilised for the creation of primitive building structures. Combination algorithms have been generated to fill missing data to create complete data sets for model mesh extraction by utilising OSM data. This data is parse through a custom model creation algorithm specifically designed for use within the runtime system. The runtime system utilises a novel Array

Indexed Render Function data structure. To organise this data, a novel scene graph data structure is utilised to improve update and rendering speeds within the runtime system. Search techniques are also employed upon the scenegraph capable of retrieving objects which satisfy user generated procedural queries within a pool of 405 building models in than 0.14099 seconds.

Aim 5, "*to implement the geospatial data rendering system*". This aim is satisfied by the unique Array Indexed Shader Function structure which gives improved rendering rates with virtual scenes, and improved interactivity and scalability of simulations. The framework surrounding the AISF allows interaction with the DirectX API through novel material objects allow procedural generation of controller objects to dynamically update the spatial and visual properties of the attached object.

Aim 6, "*to develop new metrics and benchmarks to evaluate performance of the system and the realism of the resulting 3D scenes*". This aim has been partially met. Our evaluation techniques to determine the scalability of the algorithms and data sets proved promising, but we believe that further research is needed to determine the successfulness of all components of the framework. To do this, case studies are needed and user groups are needed. This includes ethical implications and thusly should be carried out for future work.

We also reiterate the contributions of the project.

Utilising an Indexed Array Shader Functions into the domain of GIS to improve rendering speeds over commonly used Branching Shader Functions.

The research conducted into the creation of algorithms and methods for the detection and reduction of errors within OS, LiDAR, and OSM datasets. The combination of these dataset have also contributed to the creation of complete datasets which procedural generation techniques have been applied for the creation of 3D model assets. The extraction of 3D model meshes from rigid grid structures of LiDAR

The utilisation of OSM boundary polygons for the extraction of 3D model meshes from rigid grid structures of LiDAR as published in 1.4.1, paper 2.

To improve storage for the large geospatial datasets, a novel encoding format has been developed to potentially save 66% of storage needs of LiDAR. Encoding the data into texture formats allows for importing of said files into many game engines, and rendering engines. This removes issues of creating additional parsers for the raw ASCII formatted LiDAR maps.

The use of an IVI to allow data encoding for objects within a scene and procedural generation of controllers applied to said objects.

The organization scenes are generated by the creation of a novel scenegraph structure. This allows spatial partitioning by using OS reference scheme for the base of the scenegraph structure, with object categorization for the top of the scenegraph structure. This allows quick data dispersal and retrieval through large scenes.

Additional aims of the project are to produce a working prototype capable of rendering real-world environments using GIS data, and compute game technologies. We state this this has been successfully implemented. We also state that further work is needed to improve the framework.

The future work needed is as follows.

Improved rendering of scenes is needed. Within this work, emphasis on realistic scenes has been a main focus on the work. Due to issues during the development, the realism of scenes is not as advanced as hoped for. To improve this, future work is needed to improve the realism of rendering for real world virtual scenes. This includes. Improved PCG algorithms to generated improved faceted building meshes. PCG was not the focus of this research, but an integral part of it. Utilise high quality texture assets. High quality texture assets need additional resources and potential graphic artists to develop, or buy. Deferred rendering is needed to allow further lights to be added to a scene, as well as provide a platform for future advanced shading functions which rely on the multistage processes of deferred rendering. The natural step towards this would be to remove the use of Microsoft XNA rendering pipeline, and utilise a modern advanced game engine such as Unity3D[72], or UnrealEngine4[73]. These engines provide advances culling techniques to improve rendering, and provide advanced shading techniques such as environment mapping and reflections.

Scenegraph optimisations are also needed. We have stated that the searching of nodes and assets will check the type of OSM asset against the PVSTag attached to it. If no PVSTag is attached, checking of the PVSTag Building attributes will not be checked. The initial check to see if the PVSTag is of a certain type is expensive due to the use of the .Net reflection. An improvement to this is to contain all attribute within a single object which all nodes will have. The generation of this object will set all values to zero, or null, and only set the pertinent attributes. For example, if a building asset is to be generated, a Tag object is to be applied containing all previous Tag attributes (Building, Highway, Amenity etc.). This removes the check against the type, and checks against attributes only searched for. This improves search technique flexibility, and the dynamic querying of assets by removing

[72] https://unity3d.com/
[73] https://www.unrealengine.com/what-is-unreal-engine-4

additional checks against object types. It also removes the need for specialised procedures to be created for each type.

Effect cloning became an issue with larger maps. Within the evaluation section, experiments were undertaken for 5 increasing sized maps. Issues arose with map 4 and map 5 due to out of memory exceptions. The reason for the memory exception is not the amount of model mesh data as previously thought, but duplication of the effect file. During scene initialisation, the effect object is loaded and copied to every model part within the scene. It is the duplication of the file which increases memory consumption and produces an out of memory exception.

To remove duplication means to create a single instance which all model parts within a scene will share. Sharing an effect file means the parameters of the effect need updating for each mesh being rendered. This may have incurred significant bandwidth issues with the increased data passing. Another issue is the updating of PVSMaterial parameters. The algorithms in place utilise bit-fields to only update parameters which have been modified from the previous frame. If data has changed, the data will be uploaded to the effect file. Further research is needed into the benefits between the two techniques. The first duplicates the effect file, but incurs increased memory usage. The second removes the memory consumption issue by sharing a single instance of the effect file, but will increase bandwidth between the CPU and GPU.

The project has potential to be a base platform for the research into the following areas;

- Crisis Management is an overarching domain which many research objects. A platform
- City Visualisation provides stakeholders to view real-world scenes and interact with them as they see fit. The stakeholders in question are as follows, but not limited too; fire and rescue, police, city planners, property developers.
- Traffic Simulation and Visualisation technologies can be inputted into 3D geometry rendering engines to analyse and simulate real world traffic simulations.
- Classification methodologies applied to OSM object classification. Given the attributes of objects within the OSM data base, classification techniques can be utilised to determine the type of an object to an accurate representation. Adding additional data sets can improve these classifications; data sets such as LiDAR, and OS. Procedurally generated objects of OSM buildings and boundaries provide additional input for classification techniques (building size, building type by boundary type classification).

As state within section 1.3, the scope of the project does not include big-data analyse and analytic approaches are needed for future work. As stated in the [12], as of 2012, roughly 2.5 Exabyte's of data is created a day, and doubles every 40 months. It is difficult to state how much of this data is

GIS data, but with geotagging images, and advanced LiDAR point clouds, big-data processing is needed to construct understanding of this amount of data. Given the amount of data produced in modern day, big data processing has three possible techniques; scale-up, scale-down, or scale-out. Scale-up is the addition of resources (memory, processing power etc.) on a node and apply parallel processing techniques. Scale-down the amount of resources applied to processing e.g. create specific queries applied to a set. Scale-out add additional nodes onto the system, creating a distributed cluster of processing. We state that given the GIS data sets, parallel processing of the data and algorithms would be an appropriate area of research for the datasets and algorithms of our project.

Intel's Hadoop [81] supports the processing and storage of large data sets over a distributed network of computers. Thousands of nodes are organised to processing terabytes of data. Hadoop has recently been implemented to process real-time data of Facebook[74] [82]. Further advances into big-data processing has led to MapReduce[75] [74]. Hadoop, and MapReduce can improve the querying of large datasets faster due to parallelisation techniques by distributed computing. These frameworks/applications need to be researched further in the future.

We state the Map Reduction Paradigm [73][74] should be implemented within our future work. Map reduction paradigm is the development and process of parallel distributed processing of large volumes of data. The data is grouped, sorted, and processed into key/value pairs which are distributed over multiple resources. Map Reduce is appropriate for this work and the data sets available (LiDAR, OS, and OSM). The data can be organised within key/value pairs and be processed independently of neighbouring nodes within the datasets.

We state that if the project is to be redone, the experiences and knowledge gained suggest that further research into already implemented algorithms and applications, is needed, and to build upon them. The utilisation of Epic's UnrealEngine4 would benefit the project more than using Microsoft XNA's API, or the newly implemented Monogame API.

We conclude that this project has been a moderate success and a base platform for future exciting research.

[74] https://www.facebook.com

[75] https://www.ibm.com/analytics/us/en/technology/hadoop/mapreduce/

References

[1] R. a. Falconer, J. Isaacs, D. J. Blackwood, and D. Gilmour, "Enhancing urban sustainability using 3D visualisation," *Proc. ICE - Urban Des. Plan.*, vol. 164, no. DP3, pp. 163–173, Jun. 2011.

[2] J. Isaacs, "Immersive and non immersive 3D virtual city: decision support tool for urban sustainability," *… Constr. Vol …*, vol. 16, no. January, pp. 149–159, 2011.

[3] K. FEDRA, *Sustainable Urban Transportation: a Model-Based Approach*, vol. 35, no. 5–6. Taylor&Francis Inc, 2004.

[4] P. Van Oort, "Spatial data quality: from description to application," *Publ. Geod. 60*, p. 125, 2005.

[5] Aladdin Project, "Aladdin Project." [Online]. Available: http://www.aladdinproject.org/. [Accessed: 03-May-2016].

[6] N. Adams *et al.*, "The ALADDIN Project: Intelligent Agents for Disaster Management," *Proc. first Int. Jt. Conf. Auton. agents multiagent Syst. part 3 AAMAS 02*, p. 1405, 2008.

[7] N. Jennings, "ALADDIN End of Project Report," pp. 1–34, 2011.

[8] I. Parberry, J. R. Nunn, J. Scheinberg, E. Carson, and J. Cole, "SAGE : A Simple Academic Game Engine [Extended Abstract]," in *Parberry, I., Nunn, J., Scheinberg, J., Carson, E., & Cole, J. (2007). SAGE: a simple academic game engine. Proceedings of the Second Annual Microsoft Academic Days on Game Development in Computer Science Education*, 2007, pp. 90–94.

[9] OpenStreetMap.com, "OpenStreetMap," 2014. [Online]. Available: http://www.openstreetmap.org/#map=0/-85/-142. [Accessed: 10-Jan-2017].

[10] M. Goetz and A. Zipf, "OpenStreetMap in 3D – Detailed Insights on the Current Situation in Germany," in *AGILE International Conference on Geographic Information Science*, 2012, no. 1, pp. 24–27.

[11] B. Mao, "Visualisation and generalisation of 3D City Models," KTH Royal Institute of Technology, 2011.

[12] E. Brynjolfsson, "Big Data : The Management Revolution," no. OctOBeR 2012, pp. 1–9, 2013.

[13] G. Droj, S. Suba, and a Buba, “Modern techniques for evaluation of spatial data quality,” *RevCAD – J. Geod. Cadastre*, pp. 265–272, 2010.

[14] D.G.Pringle, “Data quality in GIS,” *Maynooth University*. pp. 1–6.

[15] M. Haklay, “How good is volunteered geographical information? A comparative study of OpenStreetMap and ordnance survey datasets,” *Environ. Plan. B Plan. Des.*, vol. 37, no. 4, pp. 682–703, 2010.

[16] M. G. Friberger, J. Togelius, A. B. Cardona, M. Ermacora, A. Mousten, and M. M. Jensen, “Data Games,” in *Proceedings of the Procedural Content Generation Workshop at FDG, Foundations of Digital Games*, 2013.

[17] M. G. Friberger and J. Togelius, “Generating interesting Monopoly boards from open data,” *2012 IEEE Conf. Comput. Intell. Games*, pp. 288–295, Sep. 2012.

[18] A. Cardona, A. Hansen, J. Togelius, and M. Gustafsson, “Open Trumps, a Data Game,” *fdg2014.org*.

[19] J. Togelius and M. G. Friberger, “Bar Chart Ball, a Data Game,” *Proc. 8th Int. Conf. Found. Digit. Games (FDG 2013) fdg2013.org*, pp. 451–452, 2013.

[20] P. Jokinen, J. Tarhio, and E. Ukkonen, “A comparison of approximate string matching algorithms,” *Software—Practice Exp.*, vol. 26, no. 12, pp. 1–4, 1996.

[21] Ordnance Survey, “A guide to coordinate systems in Great Britain: An introduction to mapping coordinate systems and the use of GPS datasets with Ordnance Survey mapping.” p. 46, 2008.

[22] T. L. Saaty and L. G. Vargas, “The Analytic Network Process,” *Decis. Mak. with Anal. Netw. Process*, vol. 195, pp. 1–40, 2013.

[23] K. I. Friese, M. Herrlich, and F. E. Wolter, “Using game engines for visualization in scientific applications,” in *IFIP International Federation for Information Processing*, 2008, vol. 279, pp. 11–22.

[24] M. Kada, S. Roettger, K. Weiss, T. Ertl, and D. Fritsch, “Real-time visualisation of urban landscapes using open-source software,” *Proc. ACRS 2003 ISRS*, vol. 0, pp. 5–8, 2003.

[25] B. Mao and L. Harrie, “Methodology for the Efficient Progressive Distribution and Visualization of 3D Building Objects,” *ISPRS Int. J. Geo-Information*, vol. 5, no. 10, p. 185, 2016.

[26] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, “The triangle processor and normal vector shader,” *ACM SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 21–30, 1988.

[27] T. Harada, J. McKee, and J. C. Yang, “Forward+: Bringing Deferred Lighting to the Next Level,” *Eurographics 2012 -- Short Pap.*, vol. 0, no. 0, pp. 5–8, 2012.

[28] M. G. Chajdas, M. Mcguire, and D. Luebke, “Subpixel Reconstruction Antialiasing for Deferred Shading,” *I3D ’11 Symp. Interact. 3D Graph. Games*, pp. 15–22, 2011.

[29] E. Gunnarsson and M. Olausson, “Deferred Rendering.”

[30] J. Jimenez *et al.*, “Filtering Approaches for Real-time Anti-aliasing,” *ACM SIGGRAPH Courses*, vol. 2.3, no. 4, p. 6:1-6:329, 2011.

[31] D. Harrison, “Evaluation of Open Source Scene Graph Implementations,” in *Visualization & Virtual Reality Research Group*, 2007.

[32] J. De Baare and R. Grigg, “Data-driven game development - evaluating productivity in the current generation of game-engines,” 2015.

[33] A. Herwig and P. Paar, “Game Engines : Tools for Landscape Visualization and Planning?,” *Trends GIS Virtualization Environ. Plan. Des.*, vol. 161, no. 172, pp. 1–10, 2002.

[34] R. Ouch and B. Rouse, “Developing a driving training game on Windows mobile phone using C# and XNA,” *Proc. CGAMES’2011 USA - 16th Int. Conf. Comput. Games AI, Animat. Mobile, Interact. Multimedia, Educ. Serious Games*, pp. 254–256, 2011.

[35] O. Denninger, “Game Programming and XNA in Software Engineering Education,” in *Proceedings of Computer Games and Allied Technology (CGAT08)*, 2008, no. April.

[36] M. Herrlich, H. Holle, and R. Malaka, “Integration of cityGML and collada for high-quality geographic data visualization on the PC and Xbox 360,” in *International Conference on Entertainment Computing*, 2010.

[37] E. W. Clua, P. a Pagliosa, A. Montenegro, and L. Murta, “A Fast and Safe Framework to Prototyping Physical Worlds Using XNA and GPU,” *Physics (College. Park. Md).*, pp. 8–11, 2009.

[38] N. Robinson and M. Shapcott, “Data mining information visualisation - beyond charts and graphs,” *Proc. Sixth Int. Conf. Inf. Vis.*, pp. 577–583, 2002.

[39] W. D. Wilde and M. J. Warren, “Visualisation of critical infrastructure failure,” *Aust. Inf. Warf.*

Secur. Conf., 2008.

[40] K. Al-Kodmany, “Visualization Tools and Methods in Community Planning : From Freehand Sketches,” *J. Plan. Lit.*, vol. 17, no. 2, pp. 189–211, 2002.

[41] J. Kopylec, A. D’Amico, and J. Goodall, “Visualizing cascading failures in critical cyber infrastructures,” *IFIP Int. Fed. …*, pp. 1–16, 2010.

[42] B. P. Zeile, R. Schildwächter, T. Poesch, and P. Wettels, “Production of Virtual 3D City Models from Geodata and Visualization with 3D Game Engines . A Case Study from the UNESCO World Heritage City of Bambergs Problem Status / Starting Point,” pp. 1–8, 2004.

[43] K. Appleton, A. Lovett, G. Sünnenberg, and T. Dockerty, “Rural landscape visualisation from GIS databases: a comparison of approaches, options and problems,” *Comput. Environ. Urban Syst.*, vol. 26, no. 2–3, pp. 141–162, Mar. 2002.

[44] S. Dong and V. R. Kamat, “SMART: scalable and modular augmented reality template for rapid development of engineering visualization applications,” *Vis. Eng.*, vol. 1, no. 1, p. 1, 2013.

[45] M. P. Kwan and J. Lee, “Emergency response after 9/11: The potential of real-time 3D GIS for quick emergency response in micro-spatial environments,” *Comput. Environ. Urban Syst.*, vol. 29, no. 2, pp. 93–113, 2005.

[46] Z. Lv, X. Li, B. Zhang, W. Wang, S. Feng, and J. Hu, “Big City 3D Visual Analysis,” *arXiv Prepr. arXiv1504.01379*, pp. 11–13, 2015.

[47] R. Graham, H. McCabe, and S. Sheridan, “Pathfinding in computer games,” in *ITB Journal*, 2003, vol. 4, no. 2, pp. 1–26.

[48] X. Cui and H. Shi, “A * -based Pathfinding in Modern Computer Games,” *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 11, no. 1, pp. 125–130, 2011.

[49] D. Harabor, “Clearance-based pathfinding and hierarchical annotated a* search,” *AIGameDev.com*, pp. 1–6.

[50] R. Geraerts, “Planning short paths with clearance using explicit corridors,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1997–2004, 2010.

[51] J. Hagelbäck and S. J. Johansson, “Using Multi-agent Potential Fields in Real-time Strategy Games,” no. Aamas, pp. 631–638, 2008.

[52] S. J. Johansson and J. Hagelbäck, “The Rise of Potential Fields in Real Time Strategy Bots,”

AIIDE, vol. 8, pp. 42–47, 2008.

[53] R. Khaled, M. J. Nelson, and P. Barr, “Design metaphors for procedural content generation in games,” *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. - CHI ’13*, p. 1509, 2013.

[54] R. Bidarra and K. de Kraker, “Integrating semantics and procedural generation: key enabling factors for declarative modeling of virtual worlds,” in *Proceedings of the FOCUS K3D Conference on Semantic 3D Media and Content*, 2010.

[55] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, “Procedural modeling of buildings,” 2006, vol. 1, no. 212, pp. 614–623.

[56] R. (NVIDIA C. Geiss, “Generating complex procedural terrains using the gpu,” *GPU Gems 3*. pp. 7–37, 2007.

[57] M. Banf *et al.*, “On-Demand Creation of Procedural Cities,” *Proc. Game Entertain. Technol.*, 2010.

[58] G. Kelly and H. McCabe, “Citygen: An interactive system for procedural city generation,” *Fifth Int. Conf. …*, 2007.

[59] P. Wonka, D. Aliaga, P. Müller, and C. Vanegas, “Modeling 3D Urban Spaces using Procedural and Simulation-based Techniques,” *ACM SIGGRAPH 2011 Courses ACM*. 2011.

[60] Y. I. H. Parish and P. Müller, “Procedural Modeling of Cities,” *28th Annu. Conf. Comput. Graph. Interact. Tech.*, no. August, pp. 301–308, 2001.

[61] G. Kelly and H. Mccabe, “A Survey of Procedural Techniques for City Generation,” *ITB J.*, vol. 14, pp. 87–130, 2006.

[62] S. Greuter, J. Parker, N. Stewart, and G. Leach, “Real-time procedural generation of ‘pseudo infinite’cities,” *Proc. 1st Int. Conf. Comput. Graph. Interact. Tech. Australas. South East Asia*, pp. 87–95, 2003.

[63] A. Martinovic and L. Van Gool, “Bayesian Grammar Learning for Inverse Procedural Modeling,” *2013 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 201–208, Jun. 2013.

[64] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-Based Procedural Content Generation: A Taxonomy and Survey,” *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, Sep. 2011.

[65] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin, “Procedural Generation of Roads,”

Eurographics 2010, vol. 29, no. 2, 2010.

[66] O. Emem, F. Batuk, D. Photogrammetry, R. Sensing, and B. Istanbul, "GENERATING PRECISE AND ACCURATE 3D CITY MODELS USING PHOTOGRAMMETRIC DATA."

[67] H. K. Dhonju, "Improving 3D Models by Adding Image Information," 2012.

[68] F. Leberl *et al.*, "Point Clouds: Lidar versus 3D Vision," *Photogramm. Eng. Remote Sens.*, vol. 76, no. 10, pp. 1123–1134, 2010.

[69] A. Montoya, B. Vandeportaele, S. Lacroix, and G. Hattenberger, "Flight autonomy of micro-drone in indoor environments using lidar flash camera," *IMAV 2010, Int. Micro Air Veh. Conf. Flight Compet.*, 2010.

[70] M. J. Smith, F. F. F. Asal, and G. Priestnall, "the Use of Photogrammetry and Lidar for Landscape Roughness Estimation in Hydrodynamic Studies," *ISPRS, XXXB*, vol. 3, pp. 714–719, 2004.

[71] D. Hopkinson *et al.*, "Errors in LiDAR ground elevation and wetland vegetation height estimates C. Hopkinson," *Int. Arch. Photogramm. Remote Sensing, Spat. Inf. Sci.*, vol. 36, no. 8, pp. 108–113, 2004.

[72] N. P. Russel Stuart, *Artificial Intelligence: A Modern Approach, 3rd Edition*. Prentice Hill, 2009.

[73] J. Dean and S. Ghemawat, "Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[74] B. Y. J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," *Commun. ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[75] S. Balaji, "Waterfall vs v-model vs agile : A comparative study on SDLC," *WATEERFALL Vs V-MODEL Vs Agil. A Comp. STUDY SDLC*, vol. 2, no. 1, pp. 26–30, 2012.

[76] D. Greer and Y. Hamon, "Agile software development," *Softw. - Pract. Exp.*, vol. 41, no. 9, pp. 943–944, 2011.

[77] P. S. Taylor *et al.*, "Agile Software Development," *Eth Mtec*, vol. 1, no. 6, pp. 1–5, 2009.

[78] K. Petersen, C. Wohlin, and D. Baca, "The Waterfall Model in Large-Scale," *Prod. Softw. Process Improv. 10th Int. Conf. PROFES 2009*, pp. 386–400, 2009.

[79] D. Eberly, "Triangulation by ear clipping," *Magic Software, Inc Geom. Tools*, pp. 1–13, 2008.

[80] L. Williams, "Pyramidal parametrics," *ACM SIGGRAPH Comput. Graph.*, vol. 17, pp. 1–11,

1983.

[81] T. (Tom E. . White, *Hadoop : the definitive guide*. 2015.

[82] D. Borthakur *et al.*, “Apache hadoop goes realtime at Facebook,” *Proc. 2011 Int. Conf. Manag. data - SIGMOD ’11*, no. May, p. 1071, 2011.