Advanced Data Driven Visualisation for Geo-spatial Data

Anthony Jones and Dan Cornford

School of Engineering and Applied Science, Aston University, Birmingham B4 7ET, UK,
`d.cornford@aston.ac.uk`,
WWW home page: `http://www.ncrg.aston.ac.uk/~cornfosd`

Abstract. Most current 3D landscape visualisation systems either use bespoke hardware solutions, or offer a limited amount of interaction and detail when used in realtime mode. We are developing a modular, data driven 3D visualisation system that can be readily customised to specific requirements. By utilising the latest software engineering methods and bringing a dynamic data driven approach to geo-spatial data visualisation we will deliver an unparalleled level of customisation in near-photo realistic, realtime 3D landscape visualisation. In this paper we show the system framework and describe how this employs data driven techniques. In particular we discuss how data driven approaches are applied to the spatiotemporal management aspect of the application framework, and describe the advantages these convey.

1 Introduction

In this paper we describe the basis for an extensible geo-spatial landscape visualisation system that is capable of supporting near photo-realistic rendering in realtime. In particular the framework is designed to allow the type and behaviour within the application to be controlled directly using data driven approaches as well as exposing a set of interfaces which are at a very abstract level and can be implemented as plug-ins. The research brings together elements from modern software engineering, computer games programming and Geographic Information Systems (GIS).

In Section 2 we briefly review the visualisation applications that are currently available and the state of the art in software engineering, especially data driven programming. Section 3 describes the application framework which is based around a central application hub that hosts the connection between a range of services providing the basic functionality of the core system. We describe the main subsystem components and how data driven methods will be applied as part of a scene management system. Finally in Section 4 we discuss the scene and render systems in more detail. We conclude with a summary of possible extensions to the design.

2 Background

Current geospatial representation and management libraries, as well as libraries supplying additional rendering functionality (for example, OGRE[1] and Open Scene Graph[2]) and commercial applications responsible for producing computer-based renderings based on geographical information (for example, WorldPerfect[3] and LandXplorer[4]) offer either limited functionality or are very expensive and often require specialised hardware to run effectively. In contrast many modern computer games place a heavy emphasis on geospatial representation (such as Microsoft Flight Simulator 2004[5]) and these function with increasingly realistic 3D graphics on relatively modest graphics hardware. However the direct application of a game engine to the 3D visualisation of landscape data has several drawbacks. In particular much of the game logic system and Artificial Integelligence would not be appropriate in the visualisation system, and secondly the priorities of the gaming world are different from the user requirements in landscape visualisation.

Geospatial visualisation applications in the field of GIS commonly rely on a low detail data sample of a world model for any given depicted scene. Visualisation data is either extrapolated from the world model (for example, geometry may be an extrusion of a topography layer or the 3D mesh equivalent of a Digital Elevation Model), or sourced from data that is supplementary to the world model such as high resolution satellite imagery. There is little attempt to add detail or clutter in order to improve the immersive quality of the viewer; the focus of such renderings remains the underlying world model, and its emphasis is the information implied by the given data sample. Whereas modern GIS visualisation systems commonly rely on the application of photograph-based textures in order to increase visual quality, we aim to enhance user immersion through the addition of natural entropy and the augmentation of the low resolution data sets upon which GIS often rely.

Another major element of the framework's design is its use of Data-Driven Programming (DDP), which will increase the application's flexibility and extensibility through measured exposure of system functionality. In traditional Object Oriented Programming, objects are described using class definitions, which typify the state and behaviour of the modelled real-world object. Objects with shared state or behaviour are commonly organised into a hierarchy of class inheritance, where child classes exhibit the state and behaviour of their parent types. In DDP, state and behaviour are described separately from their owning objects as components. Each component represents a closely related collection of state and behaviour which together depict a single facet of overall object functionality. Each object thus becomes an aggregation of parameterised components,

[1] `http://www.ogre3d.org/`
[2] `http://www.openscenegraph.org/`
[3] `http://www.metavr.com/products/worldperfect/worldperfect.html`
[4] `http://www.landex.de/`
[5] `http://www.microsoft.com/games/flightsimulator/`

which together describe the object as a whole. By using data to describe component parameters and combinations, a class hierarchy can be defined using one or more data files, which collectively drive the logical composition of its constituent run-time objects.

The obvious benefits of DDP are increased extensibility and flexibility; the definition of new object types and behaviours when linked to a scripting language, as well as the modification and instantiation of existing ones, can all be achieved via data manipulation *without access to application code.* In the context of visualising GIS information, the use of DPP enables the user to assemble information rich virtual environments through the combination and extension of existing scene object descriptions. For example, an illustration for visual impact assessment can be quickly tailored to highlight proposed items, add vegetation, buildings and people, or simply increase foreground detail to increase user immersion.

3 The application framework

Figure 1 provides an overview of the application framework. The framework's overall design is highly modular, which in turn leads to increased flexibility and extensibility. Loosely based on the design of an object composition framework presented in [1], the framework separates overall application functionality into a number of coherent, loosely coupled responsibilities, each of which is represented in the framework by an abstract interface (illustrated via the inner octagon in Figure 1). The concrete implementation, and thus the run-time behaviour, of each subsystem may be provided by the user in the form of a dynamically linked library[6] (the outer octagon in Figure 1 denotes such concrete implementations). The behaviour of each subsystem is data driven, with further extensibility provided through the use of application plug-ins[7]. During application execution, the central framework hub performs dynamic allocation and binding of sub-system implementations to their respective interfaces; the hub also acts as an intermediating interface between the various subsystems. A brief introduction to the subsystem responsibilities follows.

Assets and Resources

Fundamental framework subsystems represent an asset[8] processing pipeline, providing an abstraction of the low-level data access and decompression capabilities, upon which a resource[9] caching sub-system resides. Data access is via a proxy

[6] That is, a .DLL file on windows systems

[7] A *plug-in* is a portion of code that is compiled into a dynamically linked library – plug-ins commonly extend application functionality via a dedicated exposed interface.

[8] In the context of the application framework, an *asset* is the optimised data file.

[9] In the context of the application framework, a *resource* is a run-time equivalent of an asset.

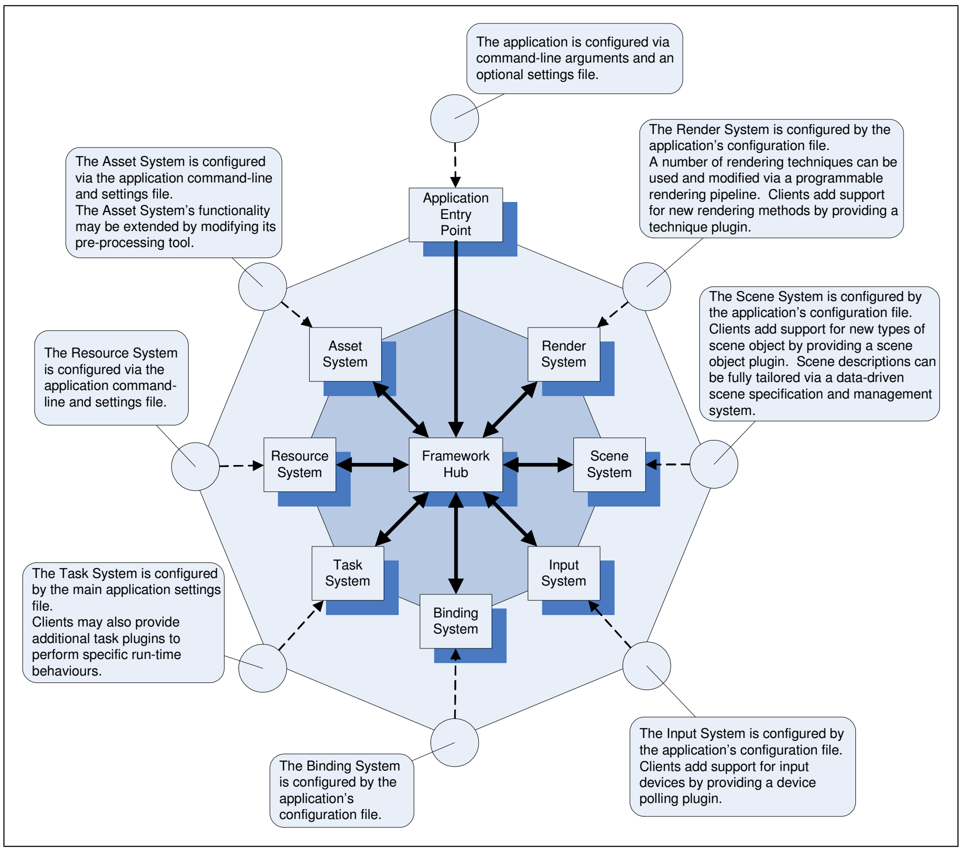
The application is configured via command-line arguments and an optional settings file.
The Asset System is configured via the application command-line and settings file. The Asset System's functionality may be extended by modifying its pre-processing tool.
The Render System is configured by the application's configuration file. A number of rendering techniques can be used and modified via a programmable rendering pipeline. Clients add support for new rendering methods by providing a technique plugin.
The Scene System is configured by the application's configuration file. Clients add support for new types of scene object by providing a scene object plugin. Scene descriptions can be fully tailored via a data-driven scene specification and management system.
The Resource System is configured via the application command-line and settings file.
Application Entry Point
Asset System
Render System
Resource System
Framework Hub
Scene System
Task System
Input System
Binding System
The Task System is configured by the main application settings file. Clients may also provide additional task plugins to perform specific run-time behaviours.
The Binding System is configured by the application's configuration file.
The Input System is configured by the application's configuration file. Clients add support for input devices by providing a device polling plugin.

Fig. 1. An overview of the application framework showing the core systems.

resource handle, which supports one of an enumerated range of basic resource types, including a binary stream, attribute tree, multi-dimensional array, and 3D mesh. A pre-processing stage performs the necessary collection and conversions from original source data to the appropriate asset data format.

Run-time Binding

In order to facilitate a data-driven methodology, the application framework includes a flexible and powerful binding system that manages associations between identifiers and memory locations via an information rich run-time type information (RTTI) system. The binding system is responsible for mapping variable identifiers and component attributes to their respective memory ranges; it is also required to manage a registry of exposed variables and functions, and to provide for type-safe, run-time bindings to them. The framework's input system, which is responsible for controlling and monitoring user input via a variety of devices, is a logical extension of the binding system. The input system uses configuration data to bind a given device plug-in, which performs device polling, to one or more run-time variables, whose values are then made available to the remainder of the system.

Application Tasks

The framework's task system manages the application's overall behaviour, which takes the form of an iteration over a number of distinct time slices[10], each consisting of a multitude of interspersed subsystem operations or events occurring in a given order. Tasks submitted to the task system are ordered and subsequently triggered according to their priority value, although triggering may be deferred until a later time slice, delayed by a given duration, or processed after an absolute time. When triggered, a task is provided with a summary of the task system's status, along with access to the framework hub and hence the application framework as a whole. The task objects themselves are either function objects or function pointers, and are both defined and supplied by subsystem implementations.

Scene Representation and Rendering

The most complex of the framework's subsystems are the scene and visualisation systems, which respectively represent and render a given virtual scene at run-time. The scene system design incorporates a component-based scene graph and dynamically adjusting spatial partitioning system (see Section 4), and will thus embody behavioural, topological and geographical properties of virtual run-time objects. The scene system will also support continuous virtual worlds by streaming scene content in and out of the simulation as required. The render system is designed to make use of the most recent release[11] of the OpenGL graphics pro-

[10] The application framework calls each time slice an *application tick*

[11] Version 2.0 at the time of writing; a specification is available here: `http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf`

gramming API, including support for both vertex and fragment shader programs. This aims to increase rendering efficiency by minimising render state changes, while maximising visual detail through the use of image-based techniques and effective lighting, shading and shadows.

4 Geospatial scene management and visualisation

Geospatial scene management

The problem of runtime representation of a given geospatial virtual scene is comprised of two sub-problems, which coincide with the topological and geometric properties of the scene. While open-source libraries are available to solve these problems (for example OGRE and Open Scene Graph), they tend to be feature rich at the expense of performance, and do not take advantage of emerging methodologies that can greatly increase an application's flexibility and extensibility. The application framework design integrates such strengths by incorporating a novel, performance oriented scene management system that extends contemporary techniques.

In every geospatial scene there exists a hierarchical topological relationship between the scene, its constituent objects, and their properties. The hierarchical relationship can be described using a tree structure that is commonly referred to as a scene graph[12], with nodes representing objects and properties, and arcs denoting the relationships between them (most notably ownership). As scene descriptions become large, the run-time efficiency of storing and manipulating them is reduced; data-driven, component-based methods [2–4] can be used to counter this. In a scene graph, objects (nodes) have properties with values, which are propagated down to the leaves unless they are overloaded en route by an equivalent property with a differing value. To reduce data duplication and its associated storage and processing overheads, scene objects can inherit from a template or archetype design, with property commonalities stored at the archetype level, and only distinctive values stored at the instance leaves. A component based design lends itself well to DDP[13], increases flexibility and extensibility as scene object inheritance hierarchies are not necessarily fixed by the software code, and is well suited to the application of design patterns such as the Abstract Factory, Composite, and Prototype patterns [5].

The problem of geometric management is commonly solved with the use of a spatial representation system, which usually takes the form of a hierarchical simplification of the scene through the use of spatial divisions in order to reduce the time complexity of spatial processing such as collision detection and frustum culling. Spatial division algorithms are well-known in the field of GIS (where they often serve an additional purpose in the linearization of a 2D or

[12] `http://en.wikipedia.org/wiki/Scene_graph`

[13] Component-centric methods actually enforce DDP as properties consist of nothing but data associations

3D representation's memory layout); examples include B-trees[14] and Octrees[15], although more advanced algorithms allow for dynamic adjusting of the dividing predicates in order to accommodate scenes containing moving objects [6].

Visualisation

While past incarnations of graphics hardware and software interfaces have utilised a fixed functionality pipeline for transform and rasterisation, today's hardware and APIs have adopted a flexible programmable pipeline that exposes aspects of the geometry and image-based processing functionality to the client. Programs written in a dedicated langauge, known as shaders, stipulate the appearance of objects in a given virtual scene by specifying tailored transform functions alongside light, material and surface characteristics.

The framework's render system will exploit such exposures by making use of shaders provided as part of data-driven scene descriptions. The resulting combination of shader technology and DDP will allow the user to customise the visual output of simulations to meet their own requirements. We anticipate a particular interest in techniques related to lighting, shading and shadows [7], [8], which have been shown to influence the sense of presence when viewing virtual environments [9].

5 Summary and Future work

In this paper we have described the basis of a data-driven geo-spatial landscape visualistion system that will be used to support realtime, near photo-realistic rendering. We have described the core design of the system, showing how data-driven programming can be used to increase its flexibility, particularly the scene system. We also introduce data-driven programming as a tool for data integration, which remains an open problem for a number of GIS applications [10].

The research is still in progress, and we intend to explore the relation between dynamic data driven approaches to GIS and visualisation solutions and the developing Geography Markup Language (GML), a Resource Description Framework like set of eXtensible Markup Language schema for describing features that exist in the real world[16]. The use of GML maps strongly to the data driven aspects of this software, and the sematic structure implied in the GML schema and well written application schema that use GML could facilitate the semi-automatic generation of visualisations using web (feature) services.

[14] http://en.wikipedia.org/wiki/B-tree

[15] http://en.wikipedia.org/wiki/Octree

[16] The GML standard and associated web feature server specifications can be found at http://www.opengeospatial.org/.

References

1. Patterson, S.: An object–composition game framework. In Treglia, D., ed.: Game Programming Gems 3. Charles River Media (2002) 15–25
2. Bilas, S.: A data-driven game object system. In: Game Developers Conference Proceedings. (2002)
3. Duran, A.: Building object systems - features, tradeoffs, and pitfalls. In: Game Developers Conference Proceedings. (2003)
4. Rene, B.: Component based object management. In Pallister, K., ed.: Game Programming Gems 5. Charles River Media (2005) 25–37
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
6. Luque, R.G., Comba, J.L.D., Freitas, C.M.D.S.: Broad-phase collision detection using semi-adjusting bsp-trees. In: SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games, New York, NY, USA, ACM Press (2005) 179–186
7. Wang, J., Sun, J.: Real-time bump mapped texture shading based-on hardware acceleration. In: VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry, New York, NY, USA, ACM Press (2004) 206–209
8. Stamminger, M., Drettakis, G.: Perspective shadow maps. In: SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (2002) 557–562
9. Mania, K., Robinson, A.: The effect of quality of rendering on user lighting impressions and presence in virtual environments. In: VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry, New York, NY, USA, ACM Press (2004) 200–205
10. Appleton, K., Lovett, A., Snnenberg, G., Dockerty, T.: Rural landscape visualisation from gis databases: a comparison of approaches, options and problems. Computers, Environment and Urban Systems **26** (2002) 141–162