12-4-2006

A Framework for Dynamic Terrain with Application in Off-road Ground Vehicle Simulations

Anthony Scott Aquilio

A FRAMEWORK FOR DYNAMIC TERRAIN WITH APPLICATION IN OFF-ROAD GROUND VEHICLE SIMULATIONS

by

ANTHONY S. AQUILIO

Under the Direction of Ying Zhu and G. Scott Owen

ABSTRACT

The dissertation develops a framework for the visualization of dynamic terrains for use in interactive real-time 3D systems. Terrain visualization techniques may be classified as either static or dynamic. Static terrain solutions simulate rigid surface types exclusively; whereas dynamic solutions can also represent non-rigid surfaces. Systems that employ a static terrain approach lack realism due to their rigid nature. Disregarding the accurate representation of terrain surface interaction is rationalized because of the inherent difficulties associated with providing runtime dynamism. Nonetheless, dynamic terrain systems are a more correct solution because they allow the terrain database to be modified at run-time for the purpose of deforming the surface. Many established techniques in terrain visualization rely on invalid assumptions and weak computational models that hinder the use of dynamic terrain. Moreover, many existing techniques do not exploit the capabilities offered by current computer hardware.

In this research, we present a component framework for terrain visualization that is useful in research, entertainment, and simulation systems. In addition, we present a novel method for deforming the terrain that can be used in real-time, interactive systems. The development of a component framework unifies disparate works under a single architecture. The high-level nature of the framework makes it flexible and adaptable for developing a variety of systems, independent of the static or dynamic nature of the solution. Currently, there are only a handful of documented deformation techniques and, in particular, none make explicit use of graphics hardware. The approach developed by

this research offloads extra work to the graphics processing unit; in an effort to alleviate the overhead associated with deforming the terrain.

Off-road ground vehicle simulation is used as an application domain to demonstrate the practical nature of the framework and the deformation technique. In order to realistically simulate terrain surface interactivity with the vehicle, the solution balances visual fidelity and speed. Accurately depicting terrain surface interactivity in off-road ground vehicle simulations improves visual realism; thereby, increasing the significance and worth of the application. Systems in academia, government, and commercial institutes can make use of the research findings to achieve the real-time display of interactive terrain surfaces.

INDEX WORDS: Terrain visualization, Dynamic terrain, Vehicle visualization, Off-road simulation

A FRAMEWORK FOR DYNAMIC TERRAIN WITH APPLICATION IN OFF-ROAD GROUND VEHICLE SIMULATIONS

by

ANTHONY S. AQUILIO

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2006

A FRAMEWORK FOR DYNAMIC TERRAIN WITH APPLICATION IN OFF-ROAD GROUND VEHICLE SIMULATIONS

by

ANTHONY S. AQUILIO

Major Professor: Ying Zhu
Major Professor: G. Scott Owen
Committee: Yi Pan
Yichuan Zhao

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2006

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1.1 Significance

Terrain is a commonplace presence in everyday life that is easily perceived, assessed, and understood. The commonality of terrain makes it an essential component in virtual worlds that seek to imitate reality. In order to be recognized and accepted by an observer, terrain visualization must meet conscious and subconscious expectations. The explicit needs may include nothing more than the visual presence of a surface element. However, for a truly immersive experience, it is necessary to provide an interactive terrain presence that conveys additional information to the user including composition and make up. Terrain visualization is an area in Computer Graphics that seeks to depict terrain in the visual display similarly to viewing its real-world counterpart.

While, the visual display of the terrain may be perceived as mere backdrop in some systems, its absence would immediately dissuade the user from viewing the scene realistically. As such, the terrain is a critical part of many simulation and visualization systems and it cannot be excluded. In addition, terrains may also be considered a universal component due to its commonality. The critical and universal nature of visualizing the terrain makes it an ideal pursuance in research.

1.2 Motivation

Currently, there exists a catalogue of algorithms with varying strengths and weaknesses in terms of algorithmic complexity and visual quality for terrain visualization. Of these algorithms, the majority of research addresses static terrains only. Methods for dynamic terrains extend upon static terrain, but also present additional, non-

trivial challenges. While the additional challenges complicate the matter, dynamic terrain solutions are more complete. Visualization and simulation systems provide more accurate results when using a dynamic terrain method; however, most systems do not employ a dynamic solution because of the challenges. With recent improvements in computer hardware, it is possible to develop practical dynamic terrain techniques that do not overtax the system and impede performance.

Since the inception of the wheel and cart, generation after generation of man has sought to improve the performance, handling, and style of ground-vehicles. In recent history, the computer has become an indispensable tool in the study of the automobile. Not only can the computer be used to design and manufacture vehicles, but it also serves as a means to study and prototype through simulation. In driving simulators and games, the user operates a virtual vehicle in a simulated world. In order to provide a good user experience, the simulation should portray driving accurately and realistically. For real world driving, the relationship that exists between the vehicle and the terrain surface defines the mobility and motion of the vehicle. In accordance, the virtual version should also take into account the relationship between the contact points and the terrain. In systems that only consider on-road driving conditions, this relationship can be assigned a fixed set of values, because on-road driving can assume the surface is rigid. Rigid roadways are characteristic of ideal driving conditions. In the real world, vehicles negotiating off-road environments may carve into the surface leaving traces remnants of tire-soil convergence. In worse-case scenarios, the soil makeup is so loose that the vehicle may become stuck and unable to escape. Driving simulators need to account for

the interactivity of the tire and soil if they are to realistically and accurately portray off-road driving.

Simulations of ground-vehicles in off-road conditions stand to benefit from inclusion of dynamic terrain solutions because it increases realism of the scene and accuracy of the experience. In an effort to improve terrain visualization systems, such as is used in ground-vehicle simulation systems, there needs to be a framework that defines the components necessary for displaying the terrain and their interrelations. In addition, workload associated with terrain visualization needs to be well-distributed; making good use of the available hardware. Integration of the framework into the system improves the application because it creates a better system through an improved user experience.

1.2.1 Motivation in Terrain Visualization

Research in terrain visualization can be justified by its universal presence and advances in the area serve to benefit a large community. While the community that makes use of terrain visualization techniques is diversified, the needs are similar. Many of these systems strive to produce highly accurate simulations and realistic visual systems that operate in real-time, further complicating the task by imposing hard deadlines to achieve interactive frame rates. Advances in terrain visualization are invaluable to all systems that endeavor to produce highly-realistic virtual worlds.

Entertainment Media, Training & Simulation, and Automotive Industries are just a few of the application domains that make common use of terrain visualization techniques. Entertainment media require terrain visualization in applications, such as video games, as a rudimentary component. Massively Multiplayer Online Role-Playing Games are one gaming genre that makes extensive use of terrain visualization techniques

in order to allow the player to explore vast, seamless landscapes. Geographical Information Systems (GIS) applications allow users to view varying levels of detail for a given area of the earth's surface. In some GIS applications, the user is able to track entities and survey the topology from varying angles, distances, and locations. Training & Simulation systems endeavor to display realistic, interactive terrains. Flight simulators, used to train military pilots, incorporate large landscapes that are displayed in real-time to the pilot during training. Outside of commercial ventures, academia has a vested interest in terrain visualization. A number of research efforts depict terrains in their simulation systems with a variety of objectives and needs. For many of these systems, the terrain must be well-represented, both visually and computationally. Research in terrain visualization can revolutionize the way consumers, trainees, and researchers interact with the virtual world.

The majority of terrain visualization techniques operate under the premise that the terrain is rigid. Algorithms designed for static terrains assume that the terrain's topological description is fixed, which limits interactivity and prevents surface reactivity. In contrast, dynamic terrains allow for the surface to be deformed at runtime. With the added capability to change the surface at runtime, environments can be more accurately portrayed. Terrain dynamics for soft surfaces like sand, snow, and mud can record object movement by deforming the terrain. Systems that use static terrain internally represent all earthen compositions with a maximum hardness factor, which is unrealistic. As a general solution, dynamic terrain is a more accurate approach for terrain visualization; however, it also presents a unique set of challenges.

1.2.2 Motivation in Ground Vehicle Visualization and Simulation

Ground-vehicle simulation in off-road environments is one simulation type that can benefit from the inclusion of a dynamic terrain solution. Manmade roadways are rigid and unaffected by the vehicles that make use of it, but off-road conditions do not offer a fixed standardized pliability. The terrain description, the vehicle, and the tire contribute to the overall trafficability. The trafficability directly influences trace remnants that can uniquely identify the vehicle and its travel path. For instance, a large truck driving across a snowy field will maneuver differently than a motorcycle traversing a sandy beach, and both will leave behind distinct markings. Unfortunately, most modern day systems ignore the relationship of the vehicle and the terrain, at the expense of visual correctness. In these situations, a dynamic terrain strategy can improve realism.

Visually-enhanced driving simulators seek to accurately simulate vehicle movement and display an image to the viewer that is faithful to what would be seen in real-life. Lack of a deformation strategy reflective of the underlying simulation model hinders visual realism, which leads to limited usability and applicability. Obviously, when the computational model in the simulation accounts for the tire-soil relation, the terrain deformation should be present in the visual system. For real-time systems, care must be taken to build the computational model in such a way that it is complex enough to provide accuracy, but simplified enough to execute quickly. Including a deformation solution to achieve terrain dynamics compliments the simulation model by offering visual confirmation of the computed results.

Terrain visualization is used in a number of applications; therefore, improvements in the field benefit a wide and varied audience. The ideal terrain handling system would

include the interface and functionality to interact with the surface in a realistic and meaningful manner. In an effort to reach this goal, it is necessary to focus efforts on dynamic terrain solutions. Assumed computational overhead associated with dynamic terrain is compensated by improving realism and a more faithful visual system. Simulations of off-road driving conditions are one application domain that can benefit from the inclusion of a dynamic terrain technique. With the aid of the underlying computational model, the visuals are drastically improved and the results more significant. Dynamic terrain has widespread applicability and usefulness; as such, improving the field of terrain visualization is in the best interest of many.

1.3 Research Objective

Ideally, terrain visualization would replicate terrestrial characteristics absolutely and completely; however, computers are currently incapable of simulating to that level of detail. As a middle ground, these systems must make simplifications in the strategies applied to the problem domain that limit variability at the cost of system precision. The objective of the research is to improve upon the current trends in terrain visualization by devising a terrain visualization framework that includes the option to offer terrains dynamics. Inclusion of a dynamic terrain system can produce a more convincing and accurate visual display at the cost of additional processing workload. Secondly, the research shall present a novel approach for deforming the terrain. In an effort to limit the impact on processing resources, the deformation strategy is well-distributed in such as manner that it makes good use of the capabilities of current hardware. A software module for simulating ground vehicles in off-road conditions is used as a practical demonstration, with the intent of demonstrating the research in a practical setting. Systems that

incorporate the framework and employ the deformation solution can achieve a more realistic and precise depiction of the terrain, than those without.

1.4 Challenges

Terrain visualization is a challenging topic in real-time computer graphics. At the root of the many difficulties is the immense size of the data in connection to the user's expectations of the visual display. The ideal terrain renderer is capable of displaying sprawling landscapes at an infinite resolution [1]. It renders a perfect visual presentation of the land from all vantage points and distances, just as if viewed in the physical world. For instance, a viewer would be able to examine a single blade of grass up close and to see the fine detail. Yet, at a distance the same blade should be completely indiscernible as it blends in with the rest of the surface matter. At this point in time, computer systems are unable to achieve this level of fidelity through brute force rendering of the scene. View dependent methods for terrain visualization are necessary to achieve this type of natural phenomena without overtaxing system resources. The difficulties faced in view-dependent methods are exacerbated when, in whole or in part, the terrain dataset can be modified at runtime.

In the modern world, our daily lives are subject to the use of ground vehicles as a means for transportation. The physics governing the interaction between a vehicle and the ground surface is very complex. This complexity is compounded by the fact that terrains are not composed of uniformly distributed soil composition or ground coverage. Topological changes are determined by the correlation of surface conditions with the attributes of the contact points, which normally consists of the tire description. Surface changes, like tread marks, can convey meaningful information about the vehicle and

surface properties to an observer. While the vehicle may not be visible anymore, many characteristics about it can be inferred through a visual examination and evaluation of observable trace remnants. Terrain deformation in a simulation system must ensure that the applied displacement faithfully expresses these relationships to the observer.

Whether for research or entertainment purposes, simulations are used to simulate something of interest to an observer. To meet the expectations of the observer, a simulator must be able follow alternate paths of execution. The course of execution may be affected by altering the state of the system, modifying parameterized values for simulated entities, or directly interfering with the simulation at runtime. At the same time as being reactive, many simulation systems are expected to run in real-time. A real-time system includes meeting schedules with time driven deadlines, in addition to maintaining computational correctness. While meeting the prescribed deadlines, many real-time simulation and visualization systems must be dynamic, reactive, and responsive to external stimuli. The visual systems considered in this research are for real-time rendering, so both the simulation system and the graphics system must meet hard deadlines.

Terrain visualization is a broad field comprised of numerous conundrums that must be addressed. The components that form a complete terrain visualization technique address data management, processing, and display. Terrain visualization requires coordination and cooperation between many components to maximize visual fidelity and optimize throughput. A number of popular, conventional methods in terrain visualization have recently become antiquated, while newer techniques do not explicitly address the features of a dynamic terrain solution.

1.4.1 Terrain Challenges

Terrain visualization encompasses many components that contribute to the display of the terrain in a scene. Each of these components brings its own set of challenges and difficulties that require attention. Strategies within modeling, rendering, and animation may require specialization in order to perform their duties in a dynamic system. In those cases where augmentation or specialization is required, it is necessary to overcome the hardships that would otherwise prevent the use of a dynamic terrain solution.

1.4.2 Simulation Challenges

Simulation systems attempt to model restricted views of the world that can be observed, studied, and investigated. Many ground-vehicle simulation systems over-simplify the relationship that exists between the terrain and the vehicle, which negatively impacts the usefulness of the system. Unlike the real-world, most simulations assume a single, optimal ground composition and the tire-soil interaction are commonly ignored. Eliminating these fallacies can promote a stronger simulation and visual depiction.

1.4.3 Paradigms for Hardware

Computers are complex, powerful machines that include specialized hardware with advanced capabilities. Hardware for the consumer market improves yearly and while some changes, like faster memory and increased bandwidth, are transparent to developers others require direct human intervention. For instance, current Graphics Processing Units have replaced the fixed functionality pipeline of previous generations with a fully programmable graphics pipeline. Using the programmable pipeline requires specialized code that is uploaded and executed on the hardware unit. To maximize the effectiveness

of these types of hardware, new programming paradigms must be learned, adopted, and mastered. New and innovative techniques must be devised that explicitly exploit the capabilities afforded by the non-transparent revolutions in consumer-market hardware.

1.4.4 Framework

Often methods cannot be adopted for use because they are poorly engineered. Unrealistic assumptions, inflexible designs, and rigid specifications affect the practicality and usefulness of a proposed solution. A general approach for dynamic terrain can be used in a number of applications like simulations, video games, and animations. For an authentic driving simulation, it is necessary to consider the tire-soil interactivity in the computational model and terrain deformation in the visualization module. A loose coupling between the dynamic terrain and computational model allows different components to adapt to meet the needs of the system. A component-based design allows future work to make use of only those pieces identified as necessary and appropriate. For example, research that seeks to study the visualization of bipedal movement across a terrain surface could adapt a high-level terrain visualization framework to address its specific needs. In an effort to encourage using these pieces together, the unified interface of a framework must first be established. The framework facilitates future studies that seek to validate, refine, or extend the results of this research initiative.

1.5 Methodology: Visualizing Dynamic Terrain in Off-Road Conditions

1.5.1 Criteria & Goals

First and foremost, the research seeks to improve the field of terrain visualization with innovations that promote dynamic terrain research and development. The first goal of the research is to develop a component framework that inherently supports dynamic

terrain. Secondly, it shall devise a novel method for deforming the terrain that can be used as a dynamic terrain solution in terrain visualization systems. In support of the research objectives, we shall develop an application for the visualization of off-road ground-vehicles that uses the framework and incorporates the technique to create meaningful surface deformations. To this end, the deformation strategy must support:

1. The generation of visually meaningful surface deformations based on the imposing object.
2. The offloading of added computational overhead incurred to the GPU.
3. Scalability through seamless integration with terrain visualization systems that support level of detail.

The research originates an innovative approach for dynamic terrain visualization suitable for interactive real-time systems, with applications in off-road terrain visualization and ground-vehicle simulation. The techniques facilitate the conveyance of meaningful information to the observer. The computational model in the simulation is simplified, yet practical, to achieve realistic results with acceptable performance. The visualization of the terrain is accomplished with a dynamic solution, in order to produce content that has meaning in the visual system. The work will be validated by a custom implementation.

1.5.2 Terrain Visualization

Terrain visualization requires specialized methods to achieve convincing visual displays that do not hinder runtime performance. Dynamic terrain solutions require that the terrain surface can be modified at runtime.

Tread marks, footprints, and other residual information can remain in soft terrain surfaces after the causal source has left the affected area. Surface deformation in ground-

vehicle simulation is necessary in the virtualization of tire-soil interaction and for providing traceability information. Whether the trace information is pronounced or subtle, it can prove invaluable in the communication of information. For example, a snowy field with tire impressions imparts vehicle presence and pathing information; whereas, a snow field with footprints relays a completely different record of activity and surface composition. Inclusion of surface deformations resulting from object-terrain interaction is necessary for the high-fidelity terrain visualization.

1.5.3 Vehicle Simulations

Driving simulators that do not account for tire-soil interaction are incomplete because they use a model that produces crude, overly-simplified visual interpretations of physical reality. Manmade roadways have specific properties considered ideal for driving; however, off-road terrain is composed by nature and suffers a non-uniform distribution that can range from ideal to worst-case driving conditions. Many systems assume that ideal driving conditions always exist and opt to ignore terrain composition variability.

To improve upon the features found in ground-vehicle simulations, the interaction between the vehicle and the ground surface must be faithfully represented by the visual display. Although the concept is not novel, the research initiative offers many distinctive qualities that make it unique and important.

1.5.4 Terrain Deformation

Dynamic terrains are unique because the terrain database can be modified at runtime. Soil Mechanics is a specialized field in the larger discipline of Physics that strives to explain the mechanics of ground matter. Physics-based dynamic terrain will

simulate the motion of the ground, such as soil slippage, sinkage and compression, using soil properties like saturation, permeability and granularity. In contrast, appearance-based methods lack detailed computational precision, but can achieve visually acceptable results using fabricated control parameters. In both cases, the resulting surface deformation provides the desired visual cue.

1.5.5 Hardware Specialization

While some advances in hardware promote system improvements transparently, others do not. For the most part, the 'opaque' methods require specialization in order to benefit from new hardware features. For Computer Graphics, a recent and significant change is the current transition from the fixed pipeline to a programmable one. With this change it has become necessary to alter the working mindset to gain the improvements afforded by the current GPU.

The recent evolution of the GPU affords us the opportunity to directly control powerful graphics hardware. The use of the programmable pipeline expedites geometry processing and can be utilized in order to reduce the computational workload placed on the CPU. However, to make full use of the feature it is necessary to augment traditional software development. The need to create, configure, and upload a specific shader program necessitates additional programming during system development (i.e. the benefit is not transparent). As time goes on, the programmable pipeline will become the de facto standard, but for now the field is in a state of transition and old techniques must be reformulated while new techniques should strive to use the GPU to its full potential.

1.5.6 Framework

Frameworks are used to assist in the validation, reproduction, and evolution of research. Logically, the visualization and simulation components are independently useful. However, they can be regarded as a single system with narrower scope, but greater functionality. Interconnectivity between system components may be exemplified in the form of a framework. The framework is both practical and useful for developing systems that include the terrain visualization. The individual components presented by the framework define a unified approach, interwoven into a high-level composition that facilitates research in dynamic terrain visualization.

1.6 Contribution & Results

Terrain visualization is commonplace and used in a number of visual systems. The ultimate goal for terrain visualization is to recreate terrain for virtual systems that is both realistic and natural. Currently, it is not feasible to completely imitate the natural terrain in a virtual world. However, as hardware improves terrain visualization should move towards its ultimate goal.

To facilitate greater realism, the research offers a unified approach for terrain visualization in a component framework solution. The framework promotes the use of dynamic terrain, when appropriate; in order to better simulate the real world. Specifically, the research presents a new method for performing fast and efficient terrain deformation. Off-road ground vehicle simulation is used for an example domain that can benefit from the use of the component framework and dynamic terrain methods. Many vehicle scenarios, like fishtailing and ‘peel-outs’ are commonplace in modern driving simulation, but terrain deformation and trafficability is not.

The research contributes to the subject area of terrain visualization within the field of Computer Graphics. The primary focus is to improve terrain visualization, with a demonstration in the area off-road ground vehicle simulation. The specialized implementation results of the research are suitable for adaptation in systems that desire the inclusion of a more realistic, more accurate simulation of off-road terrain conditions.

In the field of Computer Graphics, the solution improves terrain visualization through its innovative approach for interactive, real-time applications that employ a terrain visualization system. The component framework is a flexible, modular approach for the research and development of terrain visualization. Its component makeup allows it to adapt to the demands of the hosting system. In addition, the deformation strategy is a self-contained unit, which integrates well within the framework, but can also be used independently. As such, the standalone technique is suitable for use in any system as a general purpose terrain deformation solution. Dynamic terrain systems enhance the system and improve the user experience. Observers can derive information regarding soil composition and surface trafficability using the added capability of deformable terrains.

Many ground-vehicle simulations lack the visual display of tire-soil interactivity. For off-road driving simulations, the research presents a complete demonstration of visualizing realistic tire-soil interactivity. Systems that employ the research strategies are empowered to produce better driving simulations through improved runtime realism. Visual realism is enhanced because the terrain depicts impacting forces using the deformation technique. In general, this research is helpful for improving accuracy in computations of the system and for expanding the quantitative display of information to the observer.

Many features and capabilities available on system hardware require direct attention in the design and architecture of a solution. Systems run better when the available hardware is used effectively. Dissemination of workload is necessary to avoid bottlenecks that can debilitate the usefulness of an algorithm, operation, or framework. In an effort to balance the workload assumed, the research strives to use a 'best of breeds' approach that makes optimal use of the GPU. In particular, the solution offloads works to the GPU to expedite scene rendering by displacing the extra work incurred from the dynamic terrain. To our knowledge, no published research exists that attempts to levy the power of the GPU with a goal of fostering dynamic terrain visualization.

1.7 Thesis Overview

The remainder of the thesis is organized as follows:

Chapter 2 introduces the reader to the field of terrain visualization and explains it in the context of the component framework. The field of Computer Graphics is comprised of tasks in modeling, rendering, and animation. The framework further refines these categorical identifiers with information and techniques pertinent to terrain visualization. Subtopics include: data format, spatial partitioning, level of detail, paging, geometry maintenance, texturing, shading, atmospheric effects, collision detection, and motion control. The relationships and dependencies between the pieces are specified to construct a high-level architecture that can be used in the design, development, and analysis of terrain visualization systems.

Chapter 3 further examines background literature in the areas of terrain visualization that have received the most attention. Specifically, algorithms in spatial partitioning, level of detail, texturing, and terrain dynamics are examined at length.

Spatial partitioning and level of detail attempt to offer improved performance by limiting the amount of data processed without impairing the visual quality of the scene. In particular, level of detail allows for large, expansive terrain meshes to be used and is the focus of the majority of research in the field. Texturing for terrains can be unique due to the potentially large mesh, which has resulted in specialized algorithms that attempt to eliminate visual artifacts. Lastly, we present a review of terrain deformation strategies, which are an underlying requirement in developing a dynamic terrain solution. The detailed survey of these topics is used to establish a greater understanding of the field and its difficulties.

Chapter 4 discusses issues of concern in dynamic terrain and presents novel approaches for their handling. First, the issue of insufficient resolution is examined. A high-level specification and detailed technical solutions are provided in the context of a newly identified construct: Dynamically Divisible Regions. In addition, we designate the process for terrain deformation and then propose a detailed technique that abides by the process. Notably, the technique makes use of the GPU to minimize the impact to performance and is adaptable for use in a number of systems.

Chapter 5 reviews the development of an application system that uses the research in a fully functioning terrain solution. The component framework was used to design and develop a dynamic terrain solution that incorporates Dynamically Divisible Regions and the GPU-based deformation technique in a complete solution. The system was integrated into an application for the simulation and visualization of off-road ground vehicles. The practical nature of the research findings is exemplified in the application, which achieves greater realism by using a high-performance dynamic terrain solution.

Chapter 6 concludes the thesis with a review and discussion of future research directions. The conclusion provides a brief overview and restates the major key points covered. The section on future work documents potential avenues that should be pursued in the future for extending and improving the work.

2. A Component Framework for Terrain Visualization

In this chapter, we review general background for terrain visualization. The information is presented within the context of a component framework for terrain visualization. The architectural layout is comprised of the following top-level components:

- Modeling
- Model Services
- Rendering
- Animation
- Application Logic

Each component is decomposable into algorithmic groups and each group addresses specific needs within the system through the execution of particular tasks. Explicit and implicit relations between decomposable units interconnect the algorithms. The connections form bonds between components, which gives forth a unique framework configuration. The complexity of a specific framework instantiation is dependent on the internally used techniques, which makes it adaptable in terms of complexity. The framework formalizes the definition of a complete solution for systems in need of terrain visualization.

2.1 Introduction

Terrain visualization solutions are complex systems that employ a variety of techniques to create a convincing visual representation of a terrestrial surface. The selections made regarding data storage, maintenance, and processing influences the visual quality and the technical capabilities of the solution. Although they appear to act in isolation, many techniques in a terrain visualization system must be coordinated to work

together. A number of system attributes are shared that can be used to identify algorithms that support seamless integration between components. Shared attributes and cross component relationships give rise to the architecture of the component framework for terrain visualization.

Issues in terrain visualization have generated a wealth of research over the years; however, the findings seem fragmented and disjoint. The disparate nature is a result of the failure to examine the coordination of new techniques in the context of a complete terrain visualization system. The component framework presented in this chapter offers a unifying architecture that facilitates research with a new tool for design, development, and comparative analysis of systems. In particular, the framework defines relationships between components that aid in determining appropriateness and viability of use.

The first level of decomposition for the framework consists of the following units: Modeling, Model Services, Rendering, Animation, and Application Logic (Figure 1). Modeling, Rendering, and Animation are common action classifiers in the field of Computer Graphics, while Model Services and Application Logic are specialized concentrations whose purposes are to provide runtime operations and handle application domain logic, respectively.

The first-class units are further refined into algorithmic groups that unitize methods in order to achieve a goal. In some cases, multiple, distinct algorithms must be coordinated to achieve the task, which forms a composite algorithm. Whether singular or composite, a new algorithm that performs the duties is added to the pool of candidate strategies available for use in systems. Selection of the correct algorithm from the candidate pool is unique, depending on the system's needs and design. The discovery of

new algorithms leads to a larger candidate pool. For selection from the candidate pool, it is often preferred to choose a solution that runs efficiently and integrates well with other algorithms.

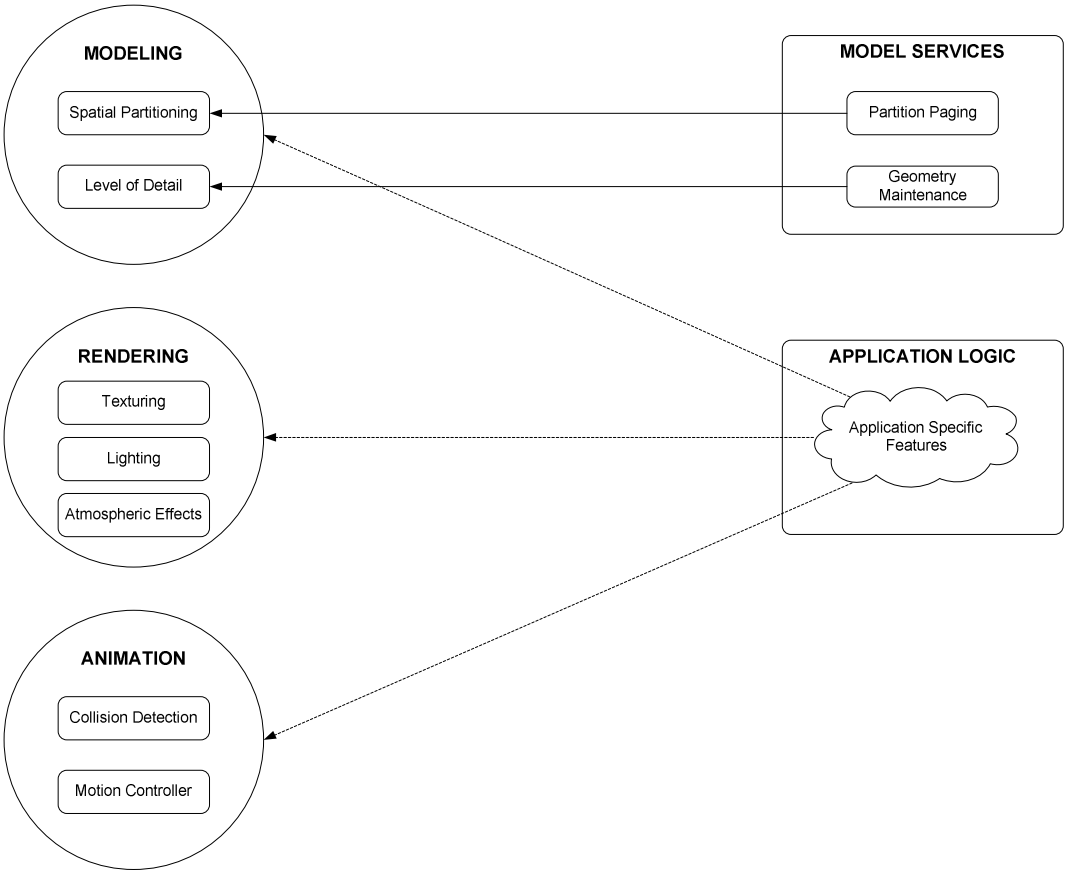


Figure 1 Component Framework Diagram for Terrain Visualization Systems.

The relationships that exist between the components and their internal algorithmic groups are fundamental to the design and implementation of a terrain solution. Lack of foresight and puzzlement over interrelations often results in problematic systems due to conflicts arising from ill-matched strategies. Constructing the system such that interrelated tasks are complimentary serves to optimize runtime performance, as well as, ease development.

2.2 Modeling

The practice of modeling is fundamental to the field of Computer Graphics. Technically, modeling involves constructing a representation of an object suitable for rendering imagery that is illustrative of the object's visual presentation. Some of the solutions used in modeling are static polygonal meshes, procedural methods, and bicubic patches. Each approach has its strengths and weakness. Choosing the correct approach is largely dependant on the intended use of the model and the requirements of the system.

Usually, a terrain mesh is represented as a polygon mesh because it is simple and versatile. A polygon mesh is composed of a well-defined set of vertices that describe the visible 'shell' of an object. A polygon mesh is decomposable into vertices and edges that describe polygonal primitives. The polygonal primitives form the virtual object's model, which is drawn to the screen that is watched by an observer. A terrain mesh is a large model and, for complex or large meshes, the decisions regarding data organization, management, and maintenance become vital to ensuring consistent frame rates.

2.2.1 Data Organization

A polygon mesh that represents a terrain surface is a simplified approximation of a topographical surface that can come from many different sources. If the model is intended to imitate an actual geographic location, then it is preferable to survey the physical terrain. Gathering sample elevation data can be used to generate an accurate geometric model. Systematically sampled elevation data provides enough vertex and edge connectivity information to create a shell approximation (i.e. the polygon mesh). In cases where the terrain is not restricted to a real world topological dataset, the method is less stringent. The first option for generating realistic, yet fictitious, terrains is with a

procedural generator, such as fractal and noise functions [2]. In systems that require more human intervention, content creators may carve the landscape in a content creation package. Regardless of the source, the end result of all production methods is a polygon mesh for use as a terrain surface in the virtual world.

Terrain meshes are notoriously large, which is the root source of many difficulties in terrain visualization. Terrains must be both expansive and detailed, creating the need to store and process vast amounts of data. Terrains sprawl to the horizon; exposing an immense amount of visible surface area. Up close, surface elements of the terrain should display high fidelity surface features to the observer. For real-time systems, there exists a definitive upper limit to the amount of information than can be processed per frame. In the context of terrain systems, the upper bound constrains the total amount of terrain data that can be processed each frame. It is necessary to limit the amount of geometry processed each frame in order to meet the deadlines of a real-time system's time constraints. However, reducing the amount of geometry creates a conflict with the need to display far-reaching, yet highly-detailed surface imagery. By seeking balance between the expansiveness of surface coverage and the intricacy of surface detail, terrain systems can maximize visual quality while minimizing resource consumption.

Data organization is of great importance to modeling in terrain visualization because data layout influences the rest of the solution. One topic of concern is the data as it relates to the underlying hardware architecture. Hardware architectures are designed to work optimally when data is organized in a specific manner. For example, operating on data and instructions with good spatial coherence and temporal locality can result in more cache hits, resulting in faster execution on modern processor architectures. For graphics

processing, data organization and the method by which the model data is submitted to the hardware will have a significant impact on performance. The second consideration when selecting a data format are its characteristic attributes and how they hinder or benefit algorithmic options. The two dominant approaches to data format and organization in polygon mesh terrain modeling are Triangulated Irregular Networks and Regular Grid Networks.

2.2.1.1 Triangulated Irregular Network

A Triangulated Irregular Network (TIN) is a tessellation of non-overlapping triangles that share edges between no more than two faces. Each vertex of the dataset represents a unique point on the surface. Edge connectivity of the vertices forms a set of triangles. The set of triangles creates a continuous surface representative of the entire terrain. TIN meshes have two predominant qualities. The characteristic of a TIN is that it is capable of representing a surface with a minimal number of vertices. Secondly, a TIN structure can represent any possible topography; including convex, concave, and planar surfaces.

As the name states, a TIN mesh is composed as a non-uniform network of vertices that interconnect to form a continuous surface of adjacent, unique triangles. A TIN uses the minimal amount of data necessary to represent the conceived surface offering a fixed amount of detail. A polygon mesh will not perfectly model a surface because it is only an approximation, but a TIN model allows for dominant surface features to be represented at optimal positions. For example, let’s consider a range of irregularly spaced mountain peaks. The vertex data for each peak and valley would be defined at the exact position the minimum and maximum extents occur. Using the minimum amount of graph information

needed to represent the terrain surface consumes the minimum amount of memory and limits the total processing overhead. TIN meshes are lightweight in terms of memory and processing consumptiveness.

A TIN mesh can represent any continuous surface. TIN meshes are unbounded and are free to define vertices at any location in the local coordinate space, even multiple heights above a single coordinate pair of the ground plane. The freedom to place vertices anywhere allows the system to model geographical constructs, such as surface folds and hollowed out volumes. This freedom makes it possible to replicate terrain features such as cliffs, overhangs, caves, and caverns. The ability to uniformly incorporate convex and concave geometry as a part of a single, unified polygon mesh is advantageous because the entire terrain can be treated similarly.

While TIN methods are optimal in respect to memory consumption and representational flexibility, they do suffer from drawbacks. The loose coupling of neighboring triangles is a failing for TIN organization. The unstructured vertex collection of a TIN mesh ensures that there is a lack of information regarding adjacent triangles because vertices are kept in a hodgepodge collection. The vertices are the endpoints of edges that define the triangulated system of polygons. Without a structured relationship between neighboring polygons, other operations on the terrain are impacted. For example, a collision detection algorithm such as an intersection test will have to test each polygon individually to determine if and where an intersection occurs. Consequently, every polygon of a mesh will be examined in the case when there is no collision. Another side effect of the polygonal disjointedness is that it can break spatial and temporal cohesiveness in current memory of system hardware. The data describing neighboring

polygons may be in separate areas of physical memory, leading to cache misses. In the worst case, poor memory use may significantly degrade performance. The intrinsic disjointedness of the polygons is a detriment to the practicality of TIN data organization for terrain models.

2.2.1.2 Regular Grid Network

A Regular Grid Network (RGN) is a triangulated tessellation of a continuous surface that is formed using samples of elevation data taken at regularly spaced intervals. In the literature, the terms heightfield and height map are often used interchangeably with RGN. An RGN mesh is a two dimensional grid of elevation samples, characterized by its organization and storage of data in a matrix. A popular source of heightfield data is the United States Geological Survey (http://www.usgs.gov/), which offers Digital Elevation Model (DEM) data. DEM data is a heightfield created by surveying and sampling elevations across areas of the United States of America at a fixed, regular interval. While the use of an underlying RGN is the preference in the majority of current literature, it is not without drawbacks.

The underlying organization of data in an RGN is ideal for many reasons. An RGN mesh has a simple underlying structure. Instead of complicated graphs or sparse matrix representations, RGN data can be easily stored and manipulated as a two dimensional matrix. The simplistic data layout is easy to work with and offers technical benefits. Given the native support of block allocation of memory in many programming languages, the implementation of an RGN is simple, if not trivial. Also, runtime performance is optimal when the data exhibits good spatial and temporal locality, which can result from the allocation and use of physical memory in large blocks. Performance is

improved because the grid layout promotes excellent block transfers of data during runtime, which achieves good cache coherence. Correctly structuring the data for good cache use increases the number of cache hits and the system will exhibit better overall performance.

The underlying matrix structure also guarantees a regularity that can be exploited in the design and development of algorithms that operate on the data. Most notably, RGN layout guarantees that triangulation results in right triangles, which can behoove the algorithmic design (Figure 2). Right triangles offer a variety of special properties that can greatly benefit processing a massive triangulated surface, like a terrain mesh. Another advantage stems from the indexed vertex configuration, which prescribes inherent relationships between adjacent vertices and defines the neighboring information explicitly. The cardinality of every vertex in the data grid is easily assessed by its indices. The regularity and structured nature of an RGN mesh is favorable to the execution of services and operations in terrain visualization.

Vertex Type	Degree	Vertex Set
Corner vertex with indices $[X, X], X = X$.	2	A, I
Corner vertex with indices $[X, Y], X \neq Y$.	3	C, G
Edge Vertex	4	B, D, F, H
Interior vertex	6	E

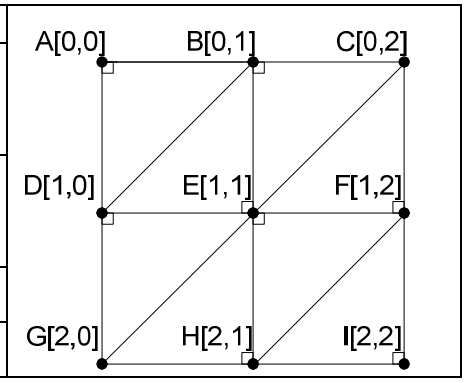


Figure 2 The rectilinear organization of data for a Regular Grid Network.

Although an RGN, and its underlying matrix format, is exceptional in many circumstances, there are constraints that can limit its applicability. The primary drawback is that not all surface features may be represented by an RGN mesh. A single RGN mesh

cannot represent multiple elevation heights across an area, whereas a TIN mesh can. As a result, features such as surface folds and hollowed-out volumes, may not be model using an RGN mesh. Consequently, this intrinsic property prevents RGN models from characterizing caves and overhangs.

Another drawback of an RGN mesh is that it tends to be inefficient with regard to memory consumption. In most cases, RGN meshes consume more memory than necessary. Every element in the matrix stores height data generated from the regular sampling of elevation information. Therefore, a block of memory large enough to store the entire range of data must is necessary regardless of the variance of the topology across the range. Unnecessary elevation data may be present where the elevation does not change noticeably. The opportunistic regularity of the sample set is also the source of wasted memory and higher primitive counts.

The last contention point with RGN meshes is the sample interval, because a poorly chosen sample interval negatively affects the final result. If the sampling occurs too infrequently, then the visual fidelity is undervalued; resulting in poor image quality. On the other hand, if sampling occurs too frequently, then the surface is over-sampled; thereby, increasing the size of the dataset and negatively affecting runtime performance. The problem faced from over-sampling is especially important for large terrains, where the dataset is already excessive. In general, the sample size can be controlled in order to establish a balance between visual fidelity and resource consumption.

The appeal of using an RGN mesh comes from the underlying matrix organization and the resulting rectangular/triangular geometric properties. These features are easily

exploited in algorithms to improve runtime capabilities. Unfortunately, the underlying matrix format also imposes rigidity, which can be burdensome.

The two predominant data organizations for terrain modeling are the Triangulated Irregular Network and the Regular Grid Network. Both approaches are useful and appropriate when used in an application that can exploit the good qualities and downplay the faults. The characteristics of data organization greatly impacts the strategies and algorithms devised and employed by the rest of the system.

2.2.2 Spatial Partitioning

In interactive, real-time 3D computer graphics applications, the workload must be completed each frame while maintaining a consistent framerate above 30 fps (frames per second). If the dataset is too large, then the time required to process it will surpass the time slice allotted to a frame, causing the system to miss deadlines and failing to run in real time. In the case of large terrain models, spatial partitioning is a strategy commonly employed to reduce the amount of geometry that gets processed in a frame. By lessening the workload, time critical deadlines are reached and the real-time criteria of the system is met.

Spatial partitioning is a technique that can be used to improve performance. With spatial partitioning, terrains systems employ a divide-and-conquer tactic when processing the geometry. Conceptually, spatial partitioning is the decomposition of a volumetric space into subspaces that corresponds to the world coordinate space. Once divided, the objects located within the bounds of the world space are sorted into the smallest subspace unit that is considered within the solution. In this way, subspace units organize the virtual objects in the scene to forms a conglomerated construct useful for improving

performance. At runtime, the organization of models into their respective partitions aid in occlusion, culling, and collision detection. Spatial portioning aids in organizing scene data, which is useful for improving performance.

Spatial subdivision techniques are either uniform or irregular. Uniform methods subdivide a space along regular subspace bounds, such that, each unit produced is of a similar geometric shape. Both the grid (2D) and the box (3D) are examples of shapes commonly used for uniform spatial subdivision. Irregular methods, on the other hand, subdivide the spatial bounds into a set of subspaces, such that, at least two subspace are dissimilar. Irregular spatial partitioning is common when defining a bounding volume for a model. For the purpose of terrain visualization, uniform methods are used almost exclusively. In many cases, the logical bounds of the world space coincide with the terrain's minimum and maximum extents. For a uniform method, the implicitly regular shape of the world bounds are subdivided into regularly shaped subspaces. The terrain geometry is subdivided into regularly shaped regions that fit into one or more subspace partitions. When possible, the subregions are defined ideally in order to fit into a partition. The end result is a disassembly of the singular terrain into pieces that can be regarded internally as individual meshes. Data organization and application requirements usually dictate the appropriateness of a given strategy. A further discussion of Spatial Partitioning is given in Section 3.2.

2.2.3 Level of Detail

Current computer architectures offer finite processing capability and software designs must compensate for the limitations imposed by a system's hardware. Real-time systems are even more overwhelming because the application must meet critical

deadlines to be considered a correct solution. Reducing the amount of data that is processed in a frame will improve an application's chances of meeting its time-critical deadlines. Level of detail (LOD) is an area of research and class of techniques that seek to reduce the computational workload by reducing the amount of data submitted for processing.

Many forms of level of detail exist in Computer Graphics, but techniques in geometric simplification algorithms are the most pertinent to the study of terrain visualization systems [3]. The basic rules driving geometric level of detail are:

1. The perceived size of a surface is directly correlated to the total number of pixels that the model surface contributes color data to.

2. The perceived size of a geometric surface is inversely proportional to its distance from the eye point.

From rules one and two it can be concluded that as an object moves away from the eye point, its visible surface will occupy fewer and fewer pixels. Without specialization, the same amount of data is processed when the object is up close as when the object is far away, even though the data is contributing less information to the image. Geometric simplification algorithms attempt to refine the set of geometry used to describe an object based on the amount of information contributed to the scene. The rational is to process only the geometry that contributes to pixel colorization and remove all excess geometry that does not.

A level of detail technique for terrain visualization is classifiable as: discrete or continuous, view-independent or view-dependent, and top-down or bottom-up. In some cases, researchers devise hybrid techniques that combine multiple algorithms in an effort to derive net benefits, but at a cost of increased algorithmic complexity. In these cases,

hybrid solutions may be decomposed, and the parts may be classification and analyzed in line with the pure approaches.

A discrete approach is typified by a two step process. The first step is to generate a series of mesh instances that vary from the original, high-detail mesh to a simplified, low-detail mesh. In most cases, the set of mesh instances is generated as a preprocessing operation during application initialization. The second step is to choose the most appropriate instance to process and display in a given frame during runtime. Discrete methods are well received because they are simple and work well with current graphics hardware. In addition, the ability to handle the bulk of computational overhead for discrete methods during initialization makes the method minimally intrusive.

In contrast to the discrete approach, continuous level of detail methods perform most, if not all, of the optimization to the model data during runtime. Typically, the model is generated to encode the complete range of mesh detail in a single instance that gets incrementally updated at runtime. The decimation and restoration of detail to the mesh is handled by the specific level of detail algorithm. Continuous methods are noted for being able to achieve optimal mesh granularity, where the object is described using the best possible subset of polygons to describe the visible surface.

The next classifier for level of detail algorithms is view-dependent or view-independent. In general, a level of detail technique that considers the viewing properties when specifying the optimal mesh is view-dependent, whereas methods that prescribe the optimal set of geometry irrespective of the view are view-independent. In practice, all methods for terrain visualization may be considered view-dependent. However, for

discrete approaches, the discrete instances are generated in a view-independent manner, but the runtime determination of which instances to display uses view-dependent criteria.

The generative simplification process for level of detail may be described as either top-down or bottom-up. A top-down approach uses a coarse, simplified mesh as a starting point and adds detail using rules prescribed by the algorithm. For hierarchical, top-down generation, the tree is built from the root downwards. In contrast, a bottom-up approach starts from the most detailed mesh and removes geometry. For hierarchical, bottom-up generation, the leaves of the tree are simplified upwards to the root. In an effort to maintain mesh likeness between successive levels of detail, both approaches strive to guarantee changes in geometry do not diverge drastically from the original input.

Even though level of detail can greatly improve runtime performance by reducing computational workload, it introduces its own set of problems that must be addressed. One common problem in discrete methods is T-Junctions, which can result in unrealistic visual artifacts. T-Junctions occur when neighboring mesh instances differ in level of detail. The variation between neighboring meshes creates inconsistencies in shared edges of adjacent polygons. T-Junctions may produce noticeable visual surface shading discontinuity and cracks at the seams between mesh instances (Figure 3a). Cracking occurs when the simplification process removes a vertex from one instance, which results in the shared edge no longer being common to both neighbors. If neighboring instances are being rendered at different levels of detail and one has removed the vertex, the displacement of the vertex may be visually perceived (Figure 3b). The last issue that may arise when using a level of detail algorithm is popping. Popping occurs when the transition between levels of detail occurs instantaneously, and the displacement of

vertices is observable. As a general rule, the presence of any of the three artifacts must be minimized, if not eradicated.

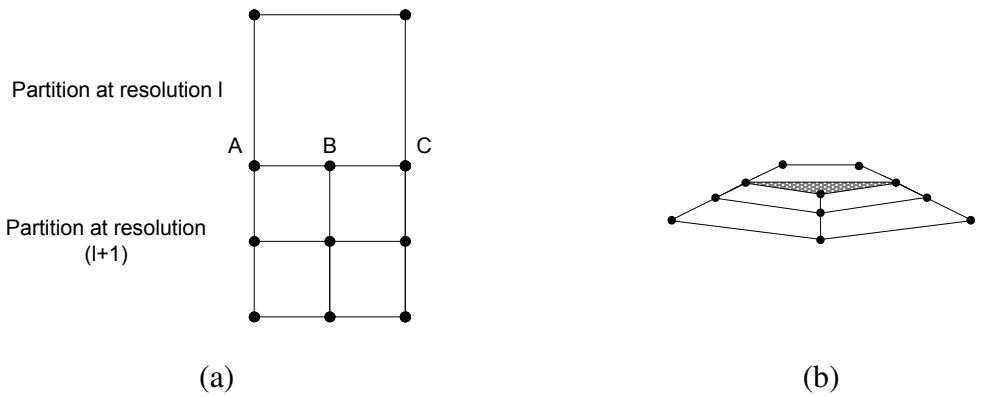


Figure 3 Cracking at a T-Junction.
T-Junction occurs because neighboring partitions are at different levels of detail. The junction occurs where the vertex B is absent from the lower resolution partitioning, but present in the higher resolution partition (a). The perspective view of the neighboring partitions in (b) reveal cracking at the seam.

Techniques for level of detail in terrain are classifiable based on the strategy used in performing the successive adaptation of geometry and the organization of data. There are five classes of techniques for level of detail in terrain visualization [4].

- *Irregular Meshes* methods are characterized by their allowance of the mesh to be maintained as a TIN. Consequently, they can achieve an optimal approximation of the surface using a minimal set of data, as has already been discussed. Decimation and reconstruction strategies are typically done on a per-polygon basis.

- *Bin-Tree Hierarchies* are characterized by the use of a tree structure to store, maintain, and mutate the mesh data. These adaptive, continuous level of detail techniques use an RGN dataset, which offers a number of optimization opportunities.

- *Bin-Tree Regions* use a tree as a reference structure to assess groups of polygons assigned to a region. Regions are not bound to a specific format, offering algorithmic flexibility to promote the development of hybrid strategies.

- *Tiled Blocks* subdivide the terrain into regions, or tiles, and then generate discrete instances of each tile at different resolutions. The complete set of non-overlapping

tiles form a mix-and-match set of possibilities to process one terrain at varying resolutions.

- *Concentric Regions* define a view-centered, hierarchy of regions. The resolution of the mesh is relative to its distance from the viewer, which creates the concentricity.

Level of detail is the largest area of research in terrain visualization. A number of algorithms exist for level of detail, because it embodies a universal need. A more detailed survey of level of detail techniques for terrain visualization is presented in Section 3.4.

The modeling component of the terrain visualization framework establishes the foundation for resource use and defines the geometric quality of the terrain mesh. Sources for terrain data may come from real world site surveys, artist invention, or procedural generation. The terrain mesh uses an underlying organization for data that serves to help or hinder the employment of related strategies. Terrain meshes are notorious for being large. Often, it is either impractical or impossible to process the entire terrain every frame. Spatial partitioning and level of detail can help reduce the amount of data that is processed without affecting image fidelity. Spatial subdivision decomposes the dataset into smaller, manageable pieces to aid in culling and expedites collision detection. Level of detail methods seek to reduce the workload by identifying and submitting a relative subset for processing each frame. Many techniques for spatial partitioning and level of detail are heavily reliant on the underlying data organization, and operate exclusively on one format or the other. Selection of complimentary techniques promotes a well integrated, standardized component for modeling tasks within a terrain visualization system.

2.3 Model Services

Modeling techniques are imperfect, and may require Model Services to compensate for the imperfections. Model Services are methods that aim to improve Modeling algorithms that have unresolved issues. In cases where the amount of data exceeds the capacity of system memory, spatial partitioning may be augmented with an integrated paging service. Many techniques for level of detail are subject to popping and cracking, which can be resolved using Geometry Maintenance services, such as geomorphs and skirts. Model Services improve Modeling solutions by extending capabilities and resolving problems.

2.3.1 Paging

For large terrains models, the amount of mesh data may exceed the amount of available in-core memory; making it impossible to load the entire dataset all at once. Paging attempts to resolve the conflict between data size and memory capacity by managing the presence of data in memory. In systems where the terrain is so large that the complete dataset cannot fit into memory, it is also unlikely that the entire terrain will be visible all at once. In these situations, only a subset of the data contributes to the fidelity of the image and, therefore, only that subset must be loaded into memory for rendering purposes. With specialization, it is possible to use spatial partitioning to generate the partitions that are then stored offline. Even though each partition is a unique set of mesh geometry, the geometric union of them is equivalent to the complete terrain mesh. At runtime, viewable partitions are paged into core memory and restored into the data structure as a spatial subunit, creating a sparsely populated entity that uses less memory without affecting the rendered scene. As the view changes, old partitions going

out of view are replaced by new partitions that come into view. In this manner, the amount of system memory required is limited to a fixed capacity but the source dataset is unbounded.

Paging is the specific technique for loading and unloading the visible partitions of terrain data. Paging limits memory consumption without infringing on the correctness of the visual presentation. At runtime, the partitions within the view are paged into memory and stored in a terrain data cache. The retrieval may be explicitly performed within the application or provided implicitly by the operating system [5]. As the view changes, different partitions come into and go out of view. When terrain partitions are needed for rendering, they are sought from the cache. When the visible partition data is in the cache, it can be immediately processed and rendered. A page fault is when the required data is not found in the cache. When a page fault occurs, the partition's data must be read into memory from an external data store. As the data is paged in, it is placed into cache memory for faster access on subsequent frames. If the cache is full, the newly paged terrain data will replace terrain data that is not needed for the scene. Effective use of paging limits the memory requirements of the terrain system solution with a minimal impact on performance.

Terrain paging can create the need to page more than just the geometry data. Vertices often have additional information associated on a per-vertex basis that is useful for generating good quality imagery. Normals, texture coordinates, vertex colors, and other attributes will need to be paged along with positional values. Texture coordinates are bound to one or more texture maps, which also implicates the necessary availability of the texture resource. The consequence of this relationship is that paging may also have

to handle loading textures and other resources in addition to the vertex data. For terrain models that use large textures, but the model does not require paging, a texture paging solution may be used exclusively [6]. Paging evokes many possible avenues for application in terrain visualization because the dataset can easily surpass the capacity of system memory.

Three properties that will impact the correctness of a terrain paging solution are: the cache size, the data access time, and the page replacement algorithm. The first concern to be addressed is the cache size because it will directly affect the cache's hit-to-miss-ratio. Increasing the size allows for more data to be stored simultaneously, improving the chances for a hit to occur, but the whole purpose of a paging system is to limit the amount of memory being used. On the other hand, if the cache size is too small, the overabundance of page faults will inhibit performance; thereby, rendering the whole solution ineffective. The second consideration is the access time for reading the terrain data from a local drive or network store. The penalty incurred will vary and is dependent on system architecture and data locality. While the hardware is outside of the scope of the algorithm, the access time can impact use of a paging algorithm. For instance, fault-intolerant algorithms are not well-placed in systems that read data over an error-prone network. The third concern is the page replacement algorithm, which determines what data is overwritten when the cache is full and new data must be stored. Preferably, the data that is overwritten is not needed again any time soon because it would have to be reloaded into the cache. Page replacement algorithms that perform poorly result in an increased number of page faults and costly accesses to external storage. For very large

terrains, paging is a necessary solution; however, it is a non-trivial task that requires careful planning and consideration to be effective.

2.3.2 Geometry Maintenance

Available processing power is wastefully consumed by an excessive amount of terrain geometry. Level of detail algorithms try to restrict the amount of geometry that must be handled in a given frame, while maintaining image fidelity. For some level of detail algorithms it is necessary to further augment the dataset to resolve issues, such as cracking and popping.

Cracks are not universal problems because they are not produced in all level of detail techniques. Specifically, discrete methods tend to display these problems while continuous methods do not. In cases where they do occur, the resolution can be handled by forcing edge vertices of partitions to match or by adding skirts. Matching edge vertices is usually incorporated directly into the algorithm. However, for solutions that do not match edge vertices natively, skirts are a general service that can be used to hide cracks by using vertical polygons around the edge of the partition [1]. Although skirts are imperfect solutions, the end result is often good enough to hide visual discrepancies in the model.

Popping is a commonplace problem that occurs with level of detail algorithms when one or more positional values are modified within a mesh between successive frames. Geomorphing is a technique that eliminates the popping effect. The instantaneous displacement of terrain vertices is hidden from the viewer by interpolating between positional values with a blend equation [7, 8]. In recent years, it was shown that Geomorphing can be performed on the GPU, which means that it is possible to minimize

computational overhead [9]. Geomorphing is very common and used in many techniques to conceal the popping effect.

Model Services offer extensions to Modeling techniques that are imperfect. Without the added features offered by the services, terrain visualization systems would suffer from reduced capability and functionality. Paging and Geometry Maintenance are two services that can improve Modeling techniques through the augmented features. Paging solutions seeks to resolve memory limitations while Geometry Maintenance techniques try to compensate for geometric changes and discontinuities. Without Model Services some techniques would be unfit for use in a terrain visualization system.

2.4 Rendering

Terrain surfaces are highly complex and, as a result, are challenging to visually replicate in Computer Graphics. In nature, terrains purvey a wealth of visual information through surface detail and environmental features that can present richness, depth, and meaning. In many cases, techniques beyond geometric representation in the model are useful for improving terrain visualization by offering greater fidelity without increasing the amount of geometry.

Whereas the terrain model defines the geometric identity of the topography, supplementary techniques can contribute to a more natural, realistic surface appearance. Rendering techniques for terrain include texturing and shading. In addition to surface rendering, terrain systems may make use of extended atmospheric models to further develop the naturalism of the terrain. Photorealistic terrain visualizations make effective use of texturing, lighting, and atmospheric effects to achieve better visual fidelity.

2.4.1 Texturing

Real world terrains are made up of many different natural elements that coexist to form the visible surface. It is unnecessary, if not impossible, to perfectly model the detailed minutia found in the soil mass of a physical terrain. The amount of geometry required to represent every element of the terrain would overload the processing pipeline, which would prevent the system from operating at interactive frame rates. In addition, many viewable surface features do not demand geometric representation.

A polygon mesh is only an approximation of the surface that defines the surface structure in terms of vertex position and edge connectivity. Other values can be associated with vertices in the polygon mesh including colors and normals. In the conversion from a continuous surface to a discrete surface, one part of terrain information that gets lost is the surface's color definition across the entire terrain. It is unreasonable to accurately reproduce a terrain surface's coloring using only vertex coloring, because only one color can be associated with each vertex of a polygon in the mesh.

Texturing is a technique for varying surface properties in an effort to imitate surface detail that is not provided in the geometry of the model [2]. Catmull produced the first images to include texture mapped models, revolutionizing the way surface detail is applied to the geometry [10]. While specialized variants of texture mapping exist, the original concept of texture mapping remains valid and underlies them all. The first step is to create a texture map, which is an image that imitates the surface properties of the model and is (usually) not present in the geometric representation. In the case of the terrain surface, elements such as blades of grass or cracks in dried dirt are good candidates for inclusion in the texture map. Next, each vertex in the face of a polygon is

assigned a texture coordinate (u, v) that correlates with a point in the texture map. The set of (u, v) coordinates assigned to the vertices of one polygon defines an area of the texture map. When the polygon is drawn to the backbuffer, an interpolated lookup supplies the rendering system with color values to assign to each fragment. In effect, the area of the texture map is applied, like a decal, to the face of the polygon. Assuming the texture image faithfully imitates surface materials; texture mapping achieves transference of surface details onto the polygon that would not be present otherwise. Texture mapping provides variations in surface appearance and polygon face colorization that can drastically improve the realism of the rendered scene.

Texture mapping is a common practice in terrain visualization for rendering detailed surfaces. The numbers of different types of terrain surfaces and items that contribute to their visual presentation are immense. In cases where texture map size is overly large, it may be necessary to employ a specialized texture management system to optimize resource use with the needs of the scene [6]. Trying to imitate the materials that make up a terrain surface through geometric modeling is both impractical and inefficient. With the polygon mesh approach for modeling, the terrain surface is only a surface approximation. Texture mapping can compensate for lost surface information by supplying surface materials that help to improve visual fidelity with minimal overhead.

2.4.2 Shading

Lighting is of great importance in the perception of a scene by the human eye. Even when surface topography does not change, a varied presence of light will greatly impact the viewed scene. Differences in the amount and type of light will influence how the brain interprets the colorization of the landscape. A great deal of research in

Computer Graphics focuses solely on attempting to faithfully reproduce realistic lighting in the virtual world. Currently, use of a complete, physically-accurate lighting function based on the Bidirectional Reflectance Distribution Function (BRDF) is unsuitable for use in real-time systems due to its complexity. However, simplified lighting equations can be used at a reduced computational cost that offers reasonably convincing shading effects.

The two shading models for calculating surface illumination are local and global. Local models compute the shading of a surface using only light from a light source, while global models also take into account light reflected between objects in the scene. Both approaches require a parameterization of the surface materials and the environment lighting. The parameterized values are used to calculate the color of each pixel in the image. Local models are simple in comparison to global models, yet the imagery produced is still very convincing. For truly realistic shading, global models such as Radiosity [11] or Ray Tracing [12] are very realistic; however, at the time of this writing, computer systems are unable to perform these methods fast enough for use in real-time systems. The shading model prescribes the computational accuracy of surface shading, which will attribute to visual fidelity.

In addition to the shading model, illumination of a scene can be either dynamic or static. Dynamic lighting allows for the orientation of the light to change over time, while static lighting assumes the light is fixed. Dynamic lighting is more difficult to handle, but more applicable as a general solution for terrain visualization because the Sun is a dynamic light source. Dynamic terrain lighting is difficult because the model's size and varying surface features makes it hard to compute effects, like shadows. Static lighting

uses a fixed light source that cannot move over time; therefore, shadowing may be computed once at startup and used for the duration of the scene. Many simplified terrain scenes do not offer dynamic lighting because the visual improvement does not justify the excess in computational complexity.

The difficulty of shading a terrain has left it a widely unexplored area, with only a handful of techniques being commonly used. One common approach for terrain shading is the use of light maps. Light maps are capable of providing reasonable shading for terrains with little computational cost and they can be dynamically updated to simulate dynamic lighting in real-time to create the illusion of the lights moving across the sky [13]. The light map can be blended with the texture map to visually imitate terrain self-shading. Another approach is to compute the horizon points for each elevation sample to produce soft shadows [14]. Unfortunately, horizon computation does not allow for dynamic geometry, which precludes it for use in dynamic terrains. Other possible avenues for shading include shadow maps [15] and shadow volumes [16], but as of this writing these techniques are too computationally expensive for use in real-time terrain visualization because the terrain model is too expansive [17]. The goal of a shading technique is to create the illusion of realistic lighting, which contributes to the overall convincement of the scene. An appropriate illumination technique will convey visual information that encourages greater submersion in the virtual world.

2.4.3 Atmospheric Effects

Many terrain systems include the simulation of atmospheric effects as a complement to the landscape. Every day, the atmosphere impacts our perception of the physical world. Virtual worlds attempting to replicate the physical world will include

environmental rendering effects that influence the visual presentation. The atmospheric component improves the scene and creates a more realistic system dynamic in the virtual world.

Whereas shading only considers the impact of light as it relates to the surface, atmospheric effects take into account the impact of particles of matter that may redirect light as it travels to the surface. Sometimes the consideration of airborne particles may be limited, or even disregarded, but for total realism the atmosphere plays a large role in the perception of a terrain. For many terrain visualizations the Sun is the primary light source. Atmospheric influences that can impact the contribution of light to the scene include:

- *Sky:* The presence and density of clouds in the sky should be accurately reflected within the scene. The intensity of incoming light is reduced when it passes through cloud cover. In most cases clouds will only influence the amount of light that reaches the terrain surface but, in some cases, the clouds may actually project shadows onto the terrain. Also, the color of the sky can vary depending on the time of day. It is possible to parameterize the atmosphere and use the information to create red, yellow or blue skies that create a tone or hue to terrain surface [18, 19].

- *Fog/Haze:* Fog is a natural phenomenon of nature. When present, airborne particles create a fogginess or haziness that reduces visibility; obscuring the perception of the terrain as it spans out into the distance. Examples include morning mist and the haze beneath the canopy of a rainforest. Both conditions produce haze that reduces visibility. In the simulation of these types of environments, the inclusion of fog and haze is needed for the correct visual depiction of the terrain and its ecosystem. Over time, fog has become a staple of mainstream computer graphics applications and hardware support is commonly available.

The rendering component serves to improve realism by adding detail without requiring additional geometry. In place of mesh geometry, techniques are performed that contribute surface information to the polygonal faces of the mesh; thereby, creating a more convincing scene. By associating texture data and surface materials with vertices, it

is possible to better imitate physical terrains. Shading is necessary to realistically simulate sunlight and moonlight illumination of the terrain. Atmospheric effects are complementary techniques that aid in further improving realism. The result from properly employing rendering methods is a more realistic rendition of the virtual world.

2.5 Animation

In visualization, virtual system dynamics and real-time interactivity are greatly valued and provide additional information to a scene, whereas immobility and rigidity are restrictive and inhibitive. Animation, in the context of real-time 3D computer graphics systems, is the area of concentration that studies the motion and interactivity of graphical objects. The interactive relationships of objects in a scene offer implicit content that the observer instinctively interprets and cognitively processes. The inclusion of motion and object dynamics is necessary to correctly portray virtual worlds where objects interact; hence, interactive applications must include object dynamics and motion.

Animation for interactive systems is unique and challenging. One source of contention for interactive systems is that the definitive motions of the objects are indeterminate at system start up. To further complicate the matter, the motions in a real-time system requires fast calculations that achieve a qualitatively, if not quantitatively, correct movement. In most graphics applications, motions are achieved by rotating, scaling, and translating vertices that make up the geometry of an object. Using linear algebra, the individual transformations can be compounded into a single transformation matrix. For the transformation of rigid body objects, the compound transformation is used for the entire object geometry. Deformable surfaces, on the other hand, require more fine-grained control of individual vertices and may necessitate unique transformations per

vertex. Another difficulty in animation is the determining the interaction between objects. The introduction of motion begets the need to detect object-object interpenetrations; otherwise, objects can pass through one another creating a surreal version of the physical world. When handled properly, the gains from animating an interactive scene will surpass the challenges.

Terrain dynamics improves realism when it includes terrain interaction. Most terrains systems are used for the purpose of allowing a virtual object, like a ground-vehicle, to traverse the surface. There are two traits necessary in a fully interactive terrain surface.

1. The terrain must provision for the detection of impact object forces.
2. The terrain must deform as a reactive motion to contact forces, in a natural and realistic manner.

In order to facilitate the needs for a fully interactive terrain surface, terrain systems will require collision detection and motion control.

2.5.1 Collision Detection

In the physical world, two objects that have unique, solid masses cannot occupy the same space. The attempt of one object to assume a space already occupied by a separate object will result in a collision. For virtual worlds, the introduction of motion begets the need to detect object collisions because objects do not have a true mass. Instead, it is up to the system to provide the means for knowing when an object is obstructed from continuing along its current path. Many systems for collision detection exist, ranging in capability, focus, and complexity. General collision detection systems offer broader coverage, but specialized solutions can offer better performance.

Collision detection for terrain systems is unique and may be subject to specialization. Depending on the data organization, the topological description may be structured in a manner that allows for assumptions to speed up calculations. Collision detection can be performed faster in systems that use spatial partitioning. The gain in runtime execution is especially important for terrains because, unlike other object collisions, intersections with the terrain are expected to occur frequently. For example, consider terrain visualization in the context of an automotive racing game. While the driver's simulated car may or may not crash into another car during the course of the race, the car will undoubtedly 'collide' with the race track. In addition, the point of contact with the terrain must be determined every frame, justifying the need for an extremely fast and accurate collision detection solution. In terrain visualization, collision detection and the subsequent intersection tests it performs require expeditious execution and quantitatively accurate results.

While collision detection for terrains may be subject to specialization, it is not necessarily unique. Intersection tests from other areas of Computer Graphics, such as Ray Tracing, can be adapted for use in terrain queries. In the general cases, intersection can be performed by searching for the polygon that describe the surface for the intersection with a ray. Fast and accurate geometric methods for detecting ray intersections are well known. For RGN data, it is possible to perform a very fast intersection test to determine the height because the orientation of the ray is known to be to the ground plane. The speed up is derived from both the fixed orientation of the ray and the ability to quickly access the possible triangles that the ray may intersect without the need to search through

the entire set. It is also possible to ascertain if and when a collision occurs with any terrain, but meshes that use an RGN data format are much faster in doing so.

Collision detection is an vital construct in the animation and motion of objects in a virtual world that seeks to imitate physical reality. Inclusion of a good solution for detecting when and where objects interpenetrate the ground is imperative in a complete terrain visualization solution.

2.5.2 Motion Control

There is a need in interactive visual systems for a Motion Control system to ensure the object(s) move in natural and convincing ways. The job of the Motion Control system is to define parameterizations, calculate results, and apply limitations that form the algorithmic description of the object's motion. In essence, the motion control embodies the model that guides the movement and, when appropriate, revises the results in an effort to produce a meaningful, convincing motion. In most literature, the motion in an animation is presented in the context of a rigid body object moving through the world. However, for terrain visualization, the Motion Control handles individual vertex movements that provide surface deformations.

In order to obtain a realistic simulation, it is essential to convincingly resemble nature. From Physics, Newton's Third Law of Motion states: *For every action there is an equal and opposite reaction*. It follows that Newton's Third Law must be faithfully reproduced to achieve a realistic simulation. As such, an interactive terrain that seeks to simulate the physical world should react to applied forces. Without an external force applying itself to a terrain the surface remains inanimate. Once an external force is applied, the terrain topology should react regardless of whether or not the reaction is

visually obvious. In many cases, the resulting surface deformation would be visually perceivable. In simulating the behavior, the terrain mesh displaces it vertices in a manner that meaningfully simulates the interacted occurrence. For motion control purposes, the types of vertex displacement are:

- *Fixed*: The surface's vertices are static and the surface does not deform.
- *Constrained*: The surface's vertices move through a limited range of motion(s) and, consequently, the surface can only show a limited range of deformations.
- *Free-form*: The surface's vertices can move through the full range of motions, in any direction and magnitude; allowing the surface to deform into any configuration. .

In terrain visualization, the majority of systems offer a fixed displacement through an implicit motion control. Fixed displacements are common in the physical world when the applied force does not overcome the reactive force. For instance, a man pushing on the ground can not send the earth out of its orbit because the force applied by the man does not overcome the gravitational forces holding the planet in orbit. While less common, constrained and free-form displacements for terrain surfaces are more accurate as a general solution to interactive terrain visualization. In the case where the man pushes down on a pile of loose top soil, the force he applies will displace the soil into a different configuration. The surface deformation can be represented properly in a system that offers a constrained or free-form Motion Control, but not when it is fixed.

Animation systems enable graphical objects to move through the scene, creating a stronger sense of realism in the virtual world. Terrain visualization that incorporates an animated surface is a more realistic and natural presentation. When forces are applied to the surface, reactive terrain deformations are created that can improve the visual quality of the scene and accuracy of the simulation.

The motion of graphical objects is important, if not mandatory, in interactive visual systems. The familiarity of motion establishes the precedent for simulated movements. In the quest to accurately portray reality, objects of mass can not interpenetrate. To prevent interpenetration, it is necessary to detect when and where two objects collide using collision detection. As objects move they usually follow well-structured rules that employ mechanical, biological, or structural constraints. For animations, a Motion Control system dictates the range of motions allowed for one or more vertices in the system. Successful development and integration of the components for Animation improves the system by offering the observer a greater sense of immersion in the virtual world.

2.6 Application Logic and Application-Specific Features

The majority of work in terrain visualization focuses on the computer graphics methods without regard for the greater system goals. In practice, Application Logic demands the acknowledgement of application-specific features. Software applications offer a service and the service entails accomplishing a goal for the system user. In many systems, visualizing and simulating the terrain is not the absolute objective. In many systems the visualization of the terrain may contribute to the overall system objective as a supportive element. Therefore, its usefulness and correctness is subject to evaluation in the context of the application domain, where it serves as a subsystem in a larger, more complex entity.

Specialized features are common in simulation, training, video game, and research applications employing a terrain visualization system. For instance, Training & Simulation applications used to prepare military personnel might expect the accurate

visualization of the terrain, but the objective of the software is to train the participants. To assist the larger system in accomplishing its goal, the terrain system may be required to offer specialized features that do not directly contribute to the terrain's realism or runtime performance. In the context of the component framework, the added feature is application specific because it does not contribute to visualization of the terrain, yet it is still an essential feature if it is to be considered complete in the application domain.

2.7 Data Flow through Components

Individually, the components of a terrain visualization system work to perform specialized functions; however, they are related through an overarching processing workflow. Components for terrain visualization require interoperability and cooperation to achieve optimal throughput, while producing high quality imagery. The sequence of information processing divulges the interdependencies and relationships of the different components as data is processed by the system. The relationships suggest fundamental congruencies in terrain visualization systems that define interconnections, which result in either a cooperative, integrated solution or a flawed, dysfunctional one. Awareness of the relationships is essential in the development of a terrain visualization system using the component framework.

In terrain visualization, the processing of data requires an execution path that encounters the various components of the framework. The sequential encounter with components induces operational relationships that benefit from coordination. Terrain mesh data is resident as a static, offline data source that can adhere to properties of a mesh type to define cooperative constraints. While a TIN mesh has less rigid rules regarding the structure of data and its geometric properties, an RGN mesh must enforce

the strict rectilinear, evenly-spaced data layout to remain valid. For each mesh type, there are a number of well-suited methods for each component that may be employed; however, there are also many algorithms that are simply not compatible. To further complicate matters, a method that is compatible with a given mesh type may be suboptimal. Lastly, the mesh type can lend itself to eased integration and interoperability. For example, RGN methods often integrate seamlessly with methods that make use of subdivisions along regular bounds, as is the case with Quad Trees and Triangle Bin-Trees. In addition to defining integrative feasibility, the mesh type will greatly impact the compatibility and optimality of techniques employed by components in the framework.

The cooperative nature of the component framework is derivative of the information flow of terrain data (Figure 4). During each frame, the terrain data is collected, refined, and processed to generate the scene's imagery. In simple systems, the processing phase may be the only one present; whereas, in advanced systems of increased complexity all three phases are employed, with the potential for each component to handle multiple subtasks. Although system complexity may differ, the process flow through the various components may be examined in a generic manner. Runtime behavior may vary depending on the presence of components and their sequence of execution, which is system specific. In such cases, the information regarding the flow is still applicable, but the technical concerns may require reevaluation.

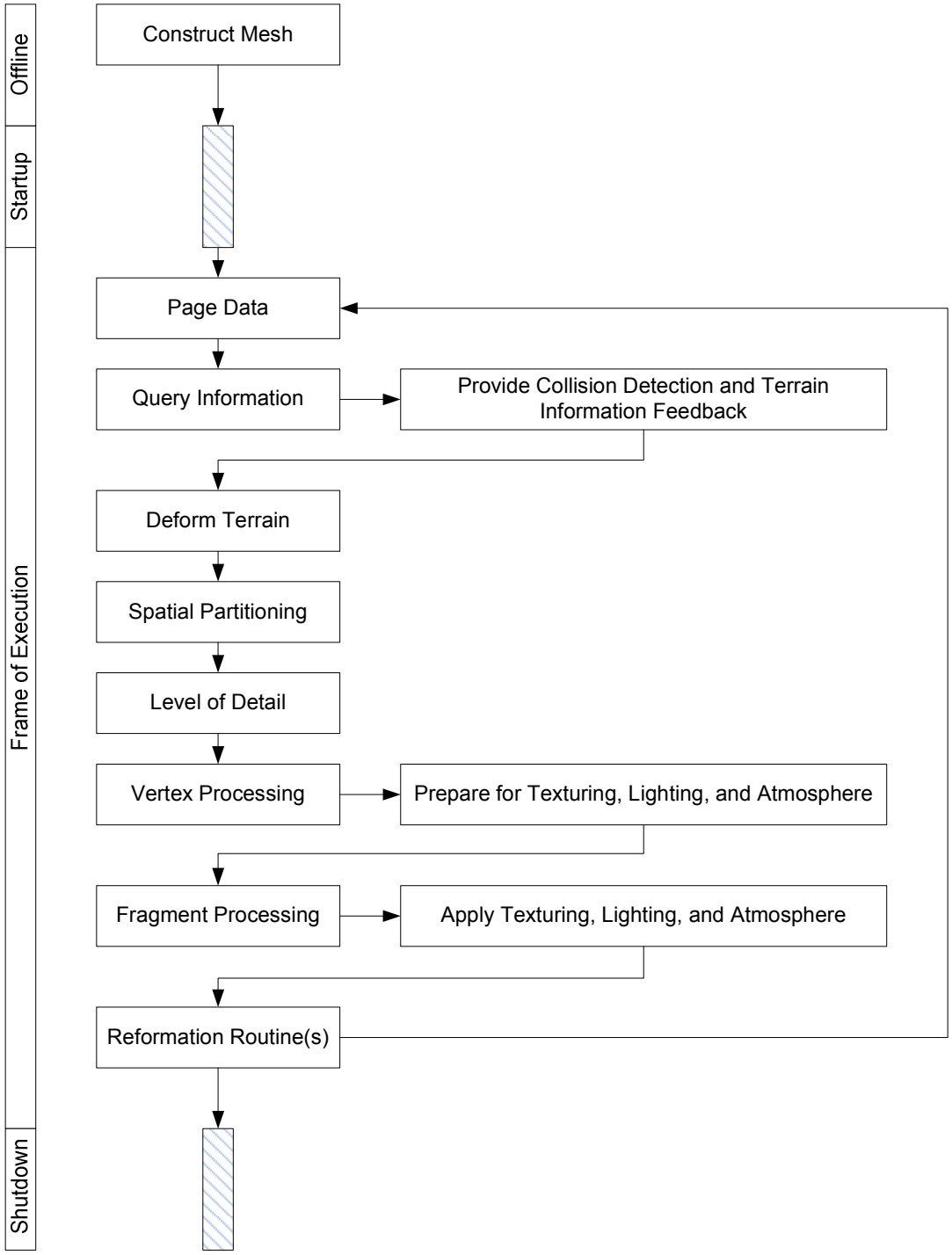


Figure 4 The Component Framework Data Flow.
Information passes through the components, defining implicit relations that can be exploited to identify interoperability issues and to improve performance.

Data is processed in terrain visualization systems to produce high quality imagery, while incurring the least computational cost. Initially, the entire terrain mesh is

assumedly going to be processed. For terrain systems supporting extremely large terrains, a paging component is invoked to ensure that all necessary terrain data is loaded into local memory from an external storage source. The source of external data may be a local disk drive or a network drive. Paging incurs latency in excess of non-paged solutions, and data may not be available immediately for processing. At this stage, it is feasible to access the terrain data for the purpose of system-required operational tasks, such as intersection tests and collision queries. Next, additional processing tasks required by dynamic terrain solutions, including surface deformation and the back propagation of these changes to remote sources, is handled. With all of the necessary data available, terrain data is processed by the spatial partitioning component. In order to improve performance, partitioning evaluates the visibility of terrain partitions and removes those deemed unnecessary to the final image from the processing queue. Next, the level of detail strategy is fed the remaining data. The multi-resolution strategy will decimate the mesh data, as a means for alleviating the total processing burden. Level of detail methods are most effective when they integrate tightly with the spatial partitioning component; thereby, allowing the two components to be executed in tandem. At this stage, the vertices are handed off to the vertex processor, which performs all of the per-vertex operations, including preparatory setup for texture mapping, shading, and atmospheric effects. Geomorphing and deformation may also be invocated during vertex processing, if employed by the solution and supported by system hardware. After vertex data processing, the fragment assembly performs the final tasks regarding texturing, shading, and atmospheric effects on a per-pixel basis. For terrain visualization systems, the finalization of fragment processing marks the end of the frame, at which point the terrain

system can prepare itself for the next frame through maintenance and reformation routines.

Each component contributes to the generation of terrain imagery in an expedient manner. Optimal flow of information through the processing pipeline mandates the use of compatible strategies that work cohesively. By considering the structural and geometric properties of the mesh, it is feasible to align component techniques in a complimentary, cohesive system. Proper coordination of components can promote optimal throughput without sacrificing image quality. The data flow is useful in coordinating the components, because it presents an abstracted view of the data path. Boundaries and overlays revealed in the data path may be used to identify integration points and compatibility issues between techniques. For instance, the culling activity in spatial partitioning and the decimation afforded in some level of detail methods can be implemented as an interwoven hybrid-solution to reduce computational overhead without affecting functionality. In contrast, these two components may be designed as disjoint features that impede the runtime performance due to poor coordination. Careful consideration of the sequence of component execution and the cooperative nature of employed techniques impacts the effectiveness of a solution in its goal to offer good performance and to generate high-quality imagery.

2.8 Review

Terrain visualization systems are a complex arrangement of seemingly unrelated pieces that must work together to create realistic terrestrial scenery. Figure 1 presents the component framework for terrain visualization systems. The system is decomposed into the following: Modeling, Model Services, Rendering, Animation, and Application Logic.

These components are further refined into specialized tasks that interoperate to produce a visually appealing, efficiently rendered landscape. Decomposing the system unveils more focused techniques that seek to achieve a goal. The techniques identified interact to produce a convincing visual interpretation of the terrain.

A well-formed, structured component framework is useful for terrain visualization. The component architecture establishes interrelations between groups and algorithms, thereby promoting a cohesive and unified approach. Designing within the framework promotes the use of cooperative and complimentary strategies, which inhibits poor design and prolonged development. The framework is one-of-a-kind in its attempt to provide an all encompassing view of terrain visualization, whereas most research focuses on a limited view of a specific technique. Also, the framework is unique in its allowance for dynamic terrain, whereas the majority of terrain solutions disregard the deformation of the terrain. Lastly, the framework makes it possible to evaluate different bodies of research in terrain visualization. Research initiatives in the area of terrain visualization are usually focused on limited subject matter. For instance, the majority of work concentrates solely on level of detail for terrains. The componentized nature of the framework can be used to classify, compare, and analyze specific techniques, algorithmic groups, and complete solutions.

3. Survey of Terrain Visualization Techniques

3.1 Introduction

There is a wealth of information in the area of terrain visualization. While the focal topics vary, they all seek to improve the field by presenting better, faster, and more elegant solutions to difficult problems. Many algorithms are devised to work as a composite or an extension of a well-known technique. In many cases, the technique used as a foundation may become outdated, resulting in the automated antiquation of its derivative works. Even though specific solutions become obsolete, the archive of theory and practices is useful and background information is always relevant. The archive provides us with a record of the various instantiations that were used over time; leading up to the modern manifestations used in current generation systems.

The intent of this chapter is to provide the reader with a survey of techniques for terrain visualization. The subject areas covered include: spatial partition, texturing, level of detail, and terrain dynamics.

3.2 Spatial Partitioning

For interactive, real-time 3D applications, all of the application logic and visualization tasks must be executed each frame at a regular, consistent rate. For visualization tasks, it may be impossible to process the entire set of geometry every frame, because the time necessary to do so will surpass the allotted time slice. One method for lessening the time it takes to process the scene is to group objects into a logical set that may be treated as a singular object. Grouped objects offer the opportunity to perform a single, preemptive operation on multiple objects in an effort to identify

which objects require further processing. For instance, grouped objects can be evaluated in an attempt to quickly cull out batches of geometry that are not visible in a given frame. In terrain visualization systems, a handful of predominant techniques are used for partitioning the terrain, and many integrated tightly with one or more level of detail techniques.

Subdivision techniques for spatial partitioning are either uniform of irregular. Uniform methods subdivide a space along regular subspace bounds, such that each unit has a similar bounding shape and, possibly, an equivalent capacity. For example, the longest edge bisection of a right isosceles triangle is an example of a uniform subdivision process. Irregular methods subdivide a space into set of subspaces, such that, at least two subspaces have unique dimensions. Although both uniform and irregular methods are possible, uniform approaches are more common because they natively support recursive and iterative methods. All of the methods covered herein are uniform; making them appropriate for terrains represented by an RGN data source.

There are many variations and specializations of uniform subdivision for the purpose of spatial partitioning. The geometry used to subdivide the space, the criteria used in split and merge operations, and the functional relationships of subspaces are all variables that allow techniques to distinguish themselves from one another. It is possible to further decompose and categorize uniform methods as hierarchical and non-hierarchical. Hierarchical methods impose an explicit relationship between subspaces in the form of parent-child and sibling relationships. In contrast, non-hierarchal methods do not offer the parent-child relations, relying solely on sibling relationships to accomplish

the tasks at hand. While hierarchal spatial subdivision schemes make exclusive use of a tree, non-hierarchical methods are not bound to any one particular data structure.

3.2.1 Non-hierarchical Methods

Non-hierarchical partitioning methods perform a single level of subdivision to generate a finite, definitive set of subspace units. The spatial partitions form a set of non-overlapping, interlocking subspaces whose geometric union is equivalent to the volumetric capacity of the world space. The most common geometric types used for the purpose of spatial partitioning are the grid in 2D and the cube in 3D; although any shape that can be tessellated into a self-similar shape is a viable alternative.

The most common non-hierarchical partitioning technique is the Uniform Grid. The Uniform Grid starts with the superspace defined by the extents of the heightfield and partitions it into regions along axially-aligned, regular bounds. The subspace regions created from the subdivision are stored in a data structure, and used for referential purposes during execution of the application. The attribute values associated with a region offer the necessary parameterization(s) for performing operations that lead to faster execution.

An object in the virtual world is associated with a region in the grid. The object is assigned to the region that it shares spatial occupancy with in the world space. For objects that are not animated, one regional assignment suffices throughout the course of execution. In contrast, animated objects require reevaluation and updated assignment to regions because they can cross boundary lines, which changes spatial occupancy. Therefore, static terrain meshes can retain the same occupancy assignment from application startup until the application stops, while dynamic terrains may require

reassignment. In many cases, dynamic terrain motion is constrained in order to ensure that deformations do not enact boundary edge transgressions. Under these circumstances, dynamic terrain meshes will not require runtime reassignment.

The Uniform Grid is a simple and straightforward technique, but is not an optimal solution. Although the Uniform Grid may meet the needs of small projects, it is not scalable and can be inefficient. The problems come from its use of the fixed size for regional subdivision. The rigid capacity forces the assignment of large objects to multiple regions, which can lead to inadvertently processing the same object multiple times. Increasing the capacity results in the assignment of more small objects to a single region; counteracting the very intention of using the technique in the first place. For projects that must support large terrains or complex virtual worlds, the lack of scalability and rigidity of the Uniform Grid may not suffice to meet the needs of the application.

3.2.2 Hierarchical Methods

Hierarchical, uniform methods perform multiple levels of subdivision to create a tree-based structure of subspaces with parent-child and sibling relations. Parent nodes correspond to the superspace for all child node subspaces. The input superspace is represented by the root and the smallest subspaces are housed by the leaf nodes. The inclusion of a structural relationship between superspaces and subspaces benefit the technique with good managerial and functional capabilities. The improved functionality comes at the cost of algorithmic complexity and memory consumption.

Hierarchical methods use the operative strategy of trees. Starting with a rectangular region, as defined by a heightfield, these methods are subdivided into self-similar spatial regions. The capacity of the space for siblings is the same for each level of

the tree. The spatial union of children in a fully-balanced branch is an improper spatial subset of the parent. Subdivision occurs until a stop condition is met. The most common stop conditions are when a minimal capacity is reached and when a maximum number of world objects are associated to the node. For systems that choose to use capacity, it is possible to guarantee that a fully-balanced tree is created; otherwise, there is potential for skewed spatial subdivision. For use with a terrain mesh, the capacity limit is more common because it correlates to a prefixed dimensional specification. For static terrains, the mesh is unchanging and assignment to a node remains unchanged during program execution. Many techniques will align the subdivision bounds with the interval of grid spacing to ensure that subregions of the terrain are assigned to only one node. Using this strategy, it is possible to construct the entire tree such that each leaf node has exactly one mesh subpart associated to it. A one-to-one relationship between the spatial nodes and the mesh geometry conjoins the algorithmic efficiency of the data structure with terrain operations.

Generally, hierarchical methods are similar in use and theory, but the different instantiations have different subdivision strategies that influence the rest of the solution. The Quad Tree is a spatial partitioning technique commonly used in terrain visualization [20]. The Quad Tree subdivides a rectangular region into four equal quadrants that can be subdivided, again, into four more quadrants ad infinitum, as shown in Figure 5.

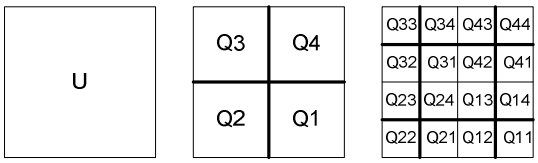


Figure 5 The first three levels of a Quad Tree.

In contrast, the Triangle Bin-Tree subdivides a right isosceles triangle using a longest edge bisection to beget two more right isosceles triangles that can be further subdivided (Figure 6). Subdivision is not limited to two dimensions. For instance, the Octree is an extrapolation of Quad Trees to three dimensions. For an Octree, the volumetric space is subdivided into cubic sub-volumes along three perpendicular planes. The added dimensionality of an Octree induces added complexity that may surpass the performance gains for scenes with simple terrain topology. Another distinguishable trait of a subdivision process is the maximum number of children for a parent node. A node in a Quad Tree can have up to four children, a Triangle Bin-Tree node can have up to two children, and an Octree node can have up to eight children. The number of children directly impacts the speed at which traversals will occur; thereby, relegating efficiency of the solution. Also, the underlying geometric shape used by the subdivision strategy tends to influence the rest of the solution. The shape for a Quad Tree is a rectangle, a Triangle Bin-Tree is a triangle, and an Octree is a rectangular volume. The specific subdivision process used differentiates spatial subdivision techniques; thereby, making each one unique.

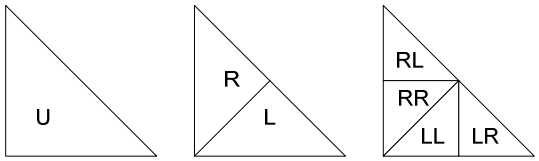


Figure 6 The first three levels of a Triangle Bin-Tree.

While hierarchical methods offer improvements over non-hierarchical methods, the underlying tree-based structure can be a performance burden. The failing for hierarchical methods is attributable to tree traversals, which are more expensive than structures that support linear indexing and direct access. As a further detriment, trees that

become skewed or have great depth(s) can exacerbate the slowdown of tree traversals. A technique for circumventing the overhead of tree traversals have been identified for the Quad Tree, but comes at the cost of maintaining specific properties, including a fully balanced tree structured with a pre-defined maximum depth [21]. Another side effect of using a tree is that the locality and coherence in system memory becomes fragmented, effecting cache use and reducing performance. Hierarchical methods offer improved logic for managing and manipulating the terrain, but the tree structure imposes additional complexity and the potential for bottlenecks.

Spatial partitioning techniques are a common occurrence in terrain visualization. Partitioning techniques subdivide the world space into a structured conglomeration of subspace, container units. Objects in the world are grouped together and bound to one or more containers. Groups of objects can be operated on as a pseudo-object, which offers the option to improve performance by reducing the total number of operations that are performed. Operations that can benefit from spatial partitioning include collision detection, culling, and geometry decimation/restoration.

3.3 Texturing

Terrain visualization makes extensive use of texture maps for presenting highly detailed surface features without the overhead of increasingly high polygon counts. Texturing is a method for varying a surface's properties from point to point, giving the appearance of surface detail that is not actually present in the surface's geometry. In lieu of creating overly complex models, one or more image maps are used as decals. The textures are superimposed onto the faces of polygons that form the mesh structure. Texture mapping interpolates the image data in determination of surface detail

colorization. The use of texturing makes it possible to present detail that would not be feasible otherwise.

Terrain visualization makes use of texturing for the purpose of showing terrain details. Terrains are composed of a number of earthen materials and natural products. Textures are useful in the display of components in nature that are too complex or computationally costly to etch into the geometric model. Earthen materials such as grass, dirt, and rock are excellent candidates for representation as textures because the equivalent geometry would be too costly to process; however, not all land features need to be abstracted from the mesh and into a texture. For instance, larger land features such as craters, boulders, and trees can be modeled as actual geometry, while grass, leaves and stones may be texture maps.

Many general practices for texture mapping are applicable to terrain visualization. However, there are only a handful of techniques that are frequently used in a many terrain visualization systems. The techniques most common for texturing the terrain mesh are: Simple Texturing, Framebuffer Compositing, Splatting, and Detail Texturing.

3.3.1 Simple Texturing

The simplest approach for terrain texturing is to use a straightforward, no-frills texturing solution that refers to one or more texture maps. In this approach a texture is applied to the terrain surface, which superimposes surface detail onto the mesh. Irregular meshes require manual texture coordinate assignment, but coordinates can be computed algorithmically for a heightfield. Given a $(n \times n)$ heightfield and a $(s \times t)$ texture map, the texture coordinates (u, v) for each vertex at indices (i, j) is computed as:

Equation 1 $$(u,v)_{i,j} = \left(i \cdot \frac{s}{n}, j \cdot \frac{t}{n} \right)$$

For effective texturing, the texture map should offer enough content to justify its use. For small terrains, a single texture may be able to provide enough texel coverage. Large terrains will require a larger texture. Increasing the texture size can offer either higher resolution detail for a restricted area or comparable detail across a greater area; however, there is an upper bound to texture size due to physical hardware limitations on system resource allocation.

For very large terrains that prefer to use a single texture across the entire terrain, it is possible to subdivide one large texture into multiple small textures. Smaller textures can be brought into and released from memory during runtime in way that enforces rules to improve memory consumption and use. A texture management system can limit the performance impact incurred from using multiple small textures [6].

Another possibility to overcome using a memory-intensive, single texture is a tiled textured. Tiled textures can be used to create a landscape texture that covers a huge terrain with a fixed size texture. Tiled textures must be seamless in order to conceal any visual artifacts that would make the tiles distinctively identifiable along edge-aligned bounds. The primary drawback of tiled textures is that repetitiveness may be perceived by a scrutinizing viewer when patterns become noticeable. The perceivable pattern can become obvious as the moiré effect becomes pronounced in the distance. In these situations, the terrain nearest the horizon will display the effect and it can be visually distracting.

In some cases, multi-texturing can offer improved surface detail. Multi-texturing uses two or more textures to create a virtual composite texture at runtime that can be applied to the terrain surface. In multi-texturing for terrains, multiple textures for the

terrain are used in a layered fashion. At runtime, a set of blending weights are used at each vertex to blend the layers into a single, cohesive texture that is applied to the terrain surface. The practice of multi-texturing is very common and can even be done on the GPU, which makes its use even more attractive. Multi-texturing is still, at its heart, simple texturing and suffers from the same drawbacks as other methods based on simple texturing. In addition to the shared problems, using more than one texture implies the consumption of additional memory, which may deter from its use in some systems.

3.3.2 Framebuffer Composition

Framebuffer Composition is a very popular technique in terrain rendering. A number of specialized implementations exist for Framebuffer Composition, but they are all derived from the same underlying theory [22-24]. Framebuffer Composition requires separate textures to represent different earthen materials, such as grass, dirt and rock (Figure 7a). In a preprocessing step, a composite, blended texture is generated and stored in video memory. The generated image appears as a banded texture with smooth cross-fades between neighboring textures. Using the sequence grass, dirt and rock; the grass texture would cross-fade into dirt texture which, in turn, would cross-fade into the rock texture (Figure 7b).

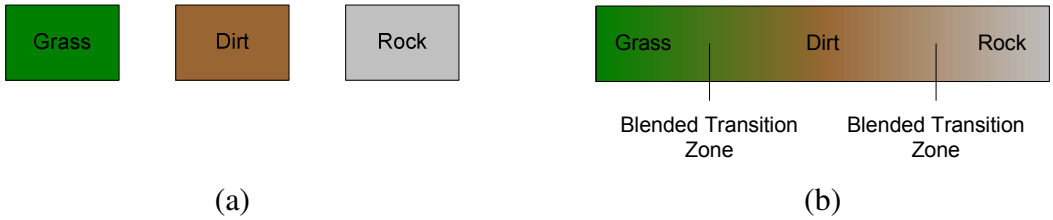


Figure 7 An example of Framebuffer Composition.
Three individual textures that represent a unique earthen material in (a) are dynamically blended to create a single texture of different earthen materials with seamless transitions between types in (b).

At runtime, each vertex in the terrain is evaluated and its texture coordinates are queried. Systems using Framebuffer Composition often perform dynamic assignment of texture coordinates to the vertices using a combination of vertex and polygon attributes. The most common attribute used is the slope of the polygonal face in conjunction with the height of the vertex. Building on the example, vertices corresponding to higher elevation and steeper face inclines would index into the texture towards the rock band while lower elevations and faces that are more parallel to the ground plane would index the texture within the grassy region of the texture. The end result is that higher, steep ground (e.g. mountain tops) displays the rock texture while lower, flat ground (e.g. knolls and fields) are textured with the grass texture.

Framebuffer composition, although a common solution, is not a perfect solution. The main problem with the technique is that it can result in visual artifacts that detract from the realism of the visualization. For instance, when using the vertex height for determining where to index into the texture, a banding effect will result from all vertices at a given height indexing into the same region of the texture. The banding effect can be diminished by adding noise, but it does not eliminate the issue. Additional problems include greater complexity and an increase in memory consumption due to the use of multiple textures. Lastly, it does not support variable levels of detail over the surface. The practical benefit with Framebuffer Composition is that it automatically produces seamless transitions between textures; however, it is not an absolute solution for terrain texturing.

3.3.3 Splatting

Splatting is another advanced technique for terrain visualization [25]. Splatting is an approach for texturing a terrain by using high-resolution, localized tiling textures

which transition nonlinearly. The original technique for Splatting uses textures that represent different earthen materials across the terrain with an irregular distribution. For instance, the rock, dirt, and grass texture can be irregularly referenced and indexed to apply variable amounts of influence to each vertex of the surface; thereby creating a unique texture through texel blending.

Splatting is similar to Framebuffer Composition, but is more robust. Splatting offers the option to additively blend various textures in an irregular distribution across the terrain surface, whereas normal Framebuffer Composition is subject to a regulated distribution. Blending between neighboring tiles is automatic, which offers the seamless transition between different textures. In this situation, tiling does not imply any sort of regularly laid out regiment for texture tiles. Instead, tiling here simply implies that one texture may be applied to more than one area of the terrain. Transition regions occur where neighboring vertices share the same set of splat textures. Weights are associated with each vertex to serve as a blending factor when computing the transition regions. The weights can be generated dynamically using an algorithmic process or assigned by a content creator to allow for more direct control. A greater sense of realism is achievable with manually assigned weights because human intervention can define distributions that closely imitate actual terrain surfaces. The ability to fine tune the surface is a strength that allows Splatting to surpass other methods in the level of realism that can be attained.

Splatting is more complex and time-consuming than other methods. With respect to Framebuffer Composition, Splatting is more complex because it requires the incorporation of a runtime system that performs a catalogue of unique operations. Additional operations include gathering polygons that share a splat texture and querying

neighbor vertices for vertex attributes. The overhead of the excess functionality inhibits the use of Splatting, because it is invasive and can impede runtime performance. In addition to complexity, the real strength of Splatting over other techniques requires human intervention, which can be time-consuming and tedious. Since changes are not automatic, manual fine-tuning of the texture weights are often required as changes are made to any one of the mesh, textures, or algorithms.

3.3.4 Detail Textures

In systems that support multi-texturing, it is common to use Detail Texturing to improve the appearance of the terrain in close vicinity of the viewer. Detail Texturing is not a self-sufficient texturing solution, but it is a complimentary approach for use in conjunction with any of the aforementioned texturing strategies. Most systems will use Simple Texturing, Framebuffer Composition, or Splatting to achieve general surface coverage. The textures used in these methods cover the breadth of the surface, but may not offer up-close details satisfactorily. If the viewer looks closely at the terrain in the immediate vicinity, pixelation can occur due to an insufficient texel to screen-pixel ratio. Detail Texturing seeks to remedy the situation, by creating highly detailed surfaces within the surrounding area of the viewer.

Detail textures are tileable textures of high-fidelity surface details. Imagery like blades of grass and cracks in dried mud are commonly used as detail textures. At runtime, the primary technique is used to supply the majority of obvious surface detail. For terrain near the viewer, the detail texture is applied as an additional layer to create the illusion of high-detail surface features on the terrain. For instance, the primary solution may apply a green texture to represent a grassy terrain, and the detail texture used is a high resolution

texture of grass blades. The detail texture is blended with the green texture to create the appearance of highly detailed grass texturing around the viewer.

Detail Texturing is effective for creating the illusion of highly detailed surfaces, without consuming exorbitant amounts of memory. To improve runtime performance, the detail texture is only blended within a localized view. Returning to the example, it is unlikely that the blades of grass would be discernible towards the horizon. As such, it is only necessary to blend with the high resolution texture in the immediate vicinity of the viewer, which is possible using view-dependent assessment of the geometry as it is processed.

As a complimentary texturing solution, detail textures are good, but are not flawless. A problem arises when the primary texturing method does not represent a surface of uniform natural composition. Typically, the detail texture represents a single, detailed earthen material, such as the blades of grass. A problem can arise in terrain areas where the primary texturing method imbues surface detail of different type. For instance, in an area of the terrain that is textured to impersonate dirt, a detail texture representing grass blades is inappropriate. The problem occurs because the detail texture applied is done in a nondiscriminatory manner. As such, the high quality detail texture of grass is inappropriately blended into the grass, dirt, rock, and transitional regions. The net result is an unconvincing, unrealistic terrain representation, because details are incorrectly correlated to the primary surface type. To combat this issue requires a smart texturing system that knows when and what detail texture to use for each element of the terrain, but this increases complexity and adds to the computational workload.

Texturing is a useful strategy for effectively visualizing terrains. While the terrain geometry is used to convey the volume and shape of the terrain, texturing provides more meticulous, meaningful visual details of the terrain composition. A number of approaches to texturing are available for terrain visualization and each one has its own strengths and weaknesses. Unfortunately, there is no best solution when evaluating the catalogue of texturing solutions because the needs, limitations, and goals vary from application to application. As with spatial partitioning, terrain texturing is greatly influenced by the level of detail technique, because of the strong relational bonds between the geometry rendered and the textures applied.

3.4 Level of detail

The largest area for research in terrain visualization has been conducted in level of detail. Interest in terrain visualization dates back to the 1970's when flight simulator research strived to improve the visual aspect of the training environment for pilots. Flight simulators display the terrain in far off distances due to the aerial view and require high fidelity imagery up close for when the aircraft is approaching the ground. In order to offer both high quality detail and expansive coverage, level of detail algorithms were, necessarily, developed for terrain visualization. The desire to meet this need has generated an abundance of information and resulted in many multi-resolution techniques for terrain rendering.

3.4.1 Background

3.4.1.1 First Generation (before 1996)

Through the mid 1990s, algorithms for level of detail in terrain rendering were primarily derivative of general approaches that could be applied to any type of mesh. The

idea behind the first generation of algorithms was to construct the surface optimally and then render the ideally triangulated terrain mesh. Methods for both continuous and discrete level of detail were proposed. First generation methods worked under the premise of limiting the throughput of triangles in order to reduce the load of graphics processing; thereby, limiting the total amount of data passing through the graphics pipeline. Reducing the workload of the graphics component improved performance because the graphics pipeline was the bottleneck. At the time, graphics acceleration hardware was not available in consumer systems and many systems used CPU-bound software rendering engines to create the visual display. Terrain rendering techniques during this period offered decent visual fidelity while performing the minimal graphics processing necessary.

3.4.1.2 Second Generation (1996 – 2001)

In 1996, a new era of algorithms for level of detail in terrain visualization begin to surface. These algorithms would drive the display with view-dependent continuous level of detail strategies that drastically improved throughput and visual fidelity. At the time, there were three solutions that dominated the field: Progressive Meshes [26], Block-based Quad Tree simplification [27], and Triangle Bin-Tree simplification[28]. In particular, Real-time Optimally Adapting Meshes (ROAM) gained acceptance and widespread popularity in terrain visualization. ROAM Using Surface Triangle Clusters (RUSTiC) is an extension to the original ROAM technique that sought to further improve performance [29]. Dynamic Extension to Resolution (DEXTER) is a general strategy that can be used with a hierarchical multi-resolution strategy, like ROAM, to offer dynamic terrain [30]. Regardless of the specifics, these solutions still sought to reduce the polygon throughput,

because consumer-level graphics acceleration hardware was very limited in its processing power and was still the bottleneck for the system.

3.4.1.3 Third Generation (2001 – Present)

From 2001 to the present, graphics acceleration hardware has improved immensely. The improvements in hardware have induced a shift in paradigms for developing graphic algorithms. Currently, the GPU can process huge amounts of data in parallel. As a result, the trend has shifted such that it is preferred to move work from the CPU to the GPU and, with this shift, the majority of existing multi-resolution algorithms for terrain visualization became deprecated. Even though the solutions themselves were outdated, the theory, insight, and knowledge gained serves as a solid foundation from which to build a new set of solutions. Some researchers have augmented old algorithms and devised hybrid solutions suitable for modern GPU architectures, while others started anew. One popular strategy is to subdivide the terrain into manageable regions, dynamically generate discrete instances for each region, and manage these instances at runtime. Other researchers would borrow ideas from texture mapping and devise geometric counterparts that could help avoid the CPU bottleneck. Some of the latest research excels even further and moves the majority of work from the CPU to the GPU. A fundamental characteristic to all of these strategies is that they do not try to build an optimal set of data, instead striving to build a ‘good enough’ data set quickly. The change in attitude and working direction has produced a number of algorithms that suitably exploit the modern GPU.

Over the years, a number of solutions for terrain visualization have been developed. Early instantiations used general purpose geometric level of detail techniques

that would be extended for use in terrain visualization systems. A number of surveys on these early algorithms are available, including [31-33]. By specifically examining the unique traits of terrain meshes, the second influx of research offered great improvements. Algorithms developed during this era have been surveyed by [34] and [35]. Second generation algorithms tended to address the generation of an optimal mesh using a variable error metric to drive the decision process. Finally, the latest incarnation of solutions for level of detail in terrains systems deserves special attention, because it is the current paradigm and most appropriate for current work in the field. The divergence from the past for these techniques is primarily derivative of the advances in hardware that has resulted in the shift of the processing bottleneck from the GPU to the CPU. As the capabilities of hardware have changed, the underlying principles behind the algorithms have also changed. Most importantly, the recent advances in graphics hardware have spurned newer techniques to opt for offloading work to the GPU.

3.4.2 Algorithmic Classes

Multi-resolution algorithms for terrain visualization seek to produce an optimal display that retains the visual fidelity of the original mesh with a lower processing cost. A number of algorithms have been proposed over the years. Although the methods work towards the same goal, they are unique in their characteristics and underlying mechanics. However, using generalized properties, it is possible to categorize methods according to an algorithmic type. Existing level of detail algorithms for terrain visualization can be classified as one of the following: Irregular Mesh, Bin-Tree Hierarchies, Bin-Tree Regions, Tiled Blocks, or Concentric Regions.

These algorithmic types serve to help unify the disparate methods under an overarching specification. While many methods exist under each category, there are one or more archetypes that act as exemplary examples of the characteristics for that family.

3.4.3 Irregular Meshes

Early algorithms in terrain visualization made use of irregular meshes to represent the terrain. The TIN layout provides an optimal surface representation, which was most suitable for systems when the graphics pipeline processing component was slow. Techniques for irregular meshes would either allow for arbitrary connectivity, such as in [32] and [36] or apply restrictions to the construction and representation of the mesh, including Delaunay triangulations as in [37] and [38]. The most commonly cited irregular mesh strategy for terrain visualization is the View Dependent Progressive Mesh [26].

3.4.3.1 Progressive Mesh

In 1997, Hoppe evolved his earlier work on Progressive Meshes (PM) [8] into View Dependent Progressive Meshes (VDPM) [26]. A Progressive Mesh is a multi-resolution technique for rendering an irregular mesh. The method uses highly detailed mesh as its input and, through a series of edge collapse operations, refines the mesh into one of lesser geometry that accurately portrays the original input mesh. During the simplification process, a record of decimations is kept that allows the mesh to be faithfully restored to its original state by performing the edge collapse's inverse operation, the vertex split. The ability to remove and restore geometry from the mesh is the fundamental level of detail feature of a Progressive Mesh. The original Progressive Mesh algorithm was used across the entire mesh, which accomplished view-independent refinement exclusively. View Dependent Progressive Meshes sought to remedy this

caveat of the original algorithm by using viewing parameters to guide the edge collapse/vertex split decisions. The original View Dependent Progressive Mesh was subject to temporal incoherence, which was resolved with the introduction of Geomorphing [39]. Geomorphing eliminates the popping effect by smoothly interpolating between successive refinements. Terrain visualization was specifically used as a demonstrative area for View Dependent Progressive Meshes, and was considered a good solution at the time.

Irregular mesh algorithms all share concepts with the Progressive Mesh in that they attempt to produce an optimal mesh that can be displayed at runtime by generating an optimal configuration of the terrain geometry. While Progressive Meshes dynamically update the mesh at runtime, some methods form the optimal polygonal decomposition of the surface offline and treat the generated instances in a discrete manner. One problem that arises from the use of an irregular mesh is that the executions of runtime operations may be slow. The irregular interrelation of primitives obligates all operations that require knowledge of an individual element to search and execute on the entire set of primitives. For example, finding the height y of the terrain at a given (x, z) coordinate requires the execution of intersection tests with each primitive in the mesh until it finds the one it intersects with. For high density, large terrains the search can be very expensive and lead to poor performance. Another problem with irregular mesh routines is that the entire set of operations for determining the optimal mesh is performed on the CPU. The current archetype for graphics programming is to offload as much work onto the GPU as is possible. In fact, recent research has shown that methods that rely on RGN data sources

can outperform TIN solutions because TIN methods are CPU-limited [40, 41]. As such, the appropriateness and applicability of irregular mesh methods has diminished.

Irregular Mesh solutions can represent a terrain surface using the least number of triangles, but do so at the cost of complexity. In order to represent the surface faithfully, vertices can be positioned and oriented in any fashion without regard for ordering or regularity. The benefit of providing an optimal primitive count is largely negated because the GPU is well-prepared to process high polygon counts quickly. Taking into consideration that the optimized mesh is constructed using per-polygon operations on the CPU further diminishes the appeal of Irregular Mesh methods. In general, the maintenance of the irregular mesh and the performance loss incurred when executing featured operations, such as collision detection, has lead to the decline in popularity of Irregular Mesh methods in current terrain visualization literature and research. However, in systems that have strict memory limitations or require folded surfaces, irregular meshes may still prove applicable.

3.4.3.2 Geomorphing

Early work by Ferguson [7] specified the use of Geomorphs in terrain visualization as a means for eliminating popping. Hoppe later applied the same theory to View Dependent Progressive Meshes and established it as a generic solution to popping in terrain visualization [39]. In the context of the visualization framework, Geomorphing is classified as a Model Service, because of its universal application domain and, therefore, it can be classified as a generic runtime service for use with many level of detail solutions in different generations and classifiers. Geomorphing integrates with a number of algorithms because it only relies on the transitional vertex distance

information to create visually seamless transitions as vertices are removed from the geometry. In order to perform Geomorphing, the distance between where the position of a vertex and the position that it implicitly move towards is needed. Fortunately, these distances are either directly available or easily calculated (Figure 8).

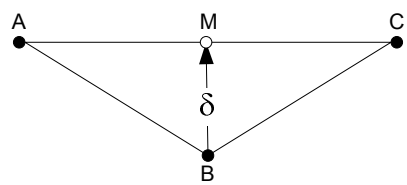


Figure 8 Midpoint displacement results in popping artifacts.
Removal of vertex B from ΔABC, implicitly moves it to the midpoint M of line segment AC.

The instantaneous switch between levels of detail results in a popping effect. Iteratively blending the vertex position from its origin to the midpoint of its two neighbors over the span of distance that exists between the two spatial locations can reduce, if not eradicate, the popping. A lookup table of distances that specify when one or more vertices are to be removed can be used when blending between levels of detail. The table stores values used in the computation of a blended elevation position. Consider the following:

Let $D_{current}$ be the current distance from a node to a triangle *T*. Also, let $D_{\max}$ equal the maximum distance that *T* may be rendered at without impacting visual fidelity. As such, the blending factor for the elevation value is computed as $BF = D_{current} \div D_{\max}$ and the blended height is computed using Equation 2.

Equation 2 $$Elevation_{current} = (1 - BF) * Elevation_{heightmap} + BF * Elevation_{midpo\,\text{int}}$$

By applying Equation 2, the movement of the camera can be used to displace vertices slowly to and from the midpoint of an edge in the parent's geometry. The

incremental movement places the vertex at the destination position before it is removed from the geometry. In this manner, the edge collapse is masked; making it non-obvious to the viewer. For modern hardware, it is even possible to perform blending entirely on the GPU; thereby offloading the additional computation required from the CPU [9].

Geomorphing is a Model Service that is commonly employed to eliminate popping. The simplicity of the algorithm and its general nature makes it appropriate for use with a number of multi-resolution solutions. The service provided disguises errant features that arise due to the decimation and restoration of terrain mesh geometry.

3.4.4 Bin-Tree Hierarchies

Bin-Tree Hierarchies are hierarchical methods that rely on an underlying RGN data source to define a fast and efficient multi-resolution strategy for terrain visualization. Methods in this class make use of a tree data structure to encode the surface's primitives; using either a Triangle Bin-Tree or Quad Tree. Bin-Tree Hierarchies offer improved performance because they can identify an optimal mesh each frame using the viewing parameters; thereby, reducing the overall amount of data processed, while still achieving good image quality.

3.4.4.1 Block-based Quad Tree

The paper by Lindstrom et al. [27] presents a technique for generating high fidelity terrain driven by a screen-space error metric through continuous level of detail refinement. The algorithm uses an RGN data source that is distributed and processed in a Quad Tree. The technique is described using a bottom-up refinement of surface geometry, but in practice it is a two-step process that performs both a top-down assessment and then a bottom-up refinement. The first step executes a block-based

simplification, while the second step refines the geometry within selected blocks on a per-vertex basis.

A block is describable as a set of elevation points assigned to the block's coverage area. The elevation points form a rectilinear grid of dimensions $2^n + 1 \times 2^n + 1, n \geq 0$ where edge vertices between neighboring blocks are shared. The root node of the Quad Tree covers the entire area and, therefore, covers the entire area of the terrain. A set of four 'sibling' blocks can be combined to create a block of lower resolution by removing every other vertex to create a new 'parent' block of the same dimension. When a vertex is removed, a new edge between two of its neighbors is formed. The midpoint of the edge corresponds to the vertex removed, and the length of the displacement from the original vertex to the new edge's midpoint is the geometric error. The geometric error is directly related to the screen-space error that is used when determining the appropriateness of further reducing the geometric complexity. The geometric error is also used to determine the mesh complexity and, upon decision of an appropriate block, the triangles within that block are iteratively examined. Vertices shared by neighboring triangles are candidates for removal. Removing a single vertex will merge two triangles into one. The error introduced by the removal of the single vertex is assessed and, if deemed appropriate, the triangles are merged (Figure 9). Newly formed triangles become candidates for further merging. The two step approach is performed each frame to build a continuous level of detail representation of the terrain surface.

Figure 9 A polygonal merge operation.
The removal of shared vertex D merges ΔABD and ΔBCD, to produce a lower resolution primitive ΔABC.

The algorithm proposed suffers from complexity and visual artifacts. The block-based approach can result in T-Junctions and cracking between neighboring blocks. In addition, dependencies on vertices shared by neighboring blocks must be enforced, which complicates the algorithm. Resolving the problems with the technique requires specialization that adds non-trivial functionality and increases overall algorithmic complexity of the solution. In addition, the specialization required can not be adapted to work in systems that require paging of streamed terrain data.

3.4.4.2 ROAMing Terrain

Duchaineau et al. published the paper *ROAMing terrain: Real-time Optimally Adapting Meshes*, which proposes a novel algorithm for continuous level of detail in terrain visualization [28]. In ROAM, a Triangle Bin-Tree is used to ensure that an optimal set of geometry is rendered during each frame by using the viewing properties. Unlike other approaches, ROAM does not suffer many of the problems associated with other multi-resolution methods, like T-Junctions and cracking. The technique is exclusively top-down and requires an RGN data source in order to produce the optimal mesh using the minimum number of triangles

The Triangle Bin-Tree is the fundamental data structure that gives the ROAM algorithm its capability. In a Triangle Bin-Tree, the root node of the tree represents a

single right triangle and, for ROAM in particular, all triangles are assumed right isosceles triangles. In order to correctly represent a regularly-spaced RGN data, two Triangle Bin-Trees are needed: one for each side of the diagonal. By performing a bisection of the longest edge from the vertex on the right angle, the triangle is recursively subdivided. An edge is created along the shortest path of the bisection to create two new similar child triangles (Figure 10). The subdivision is repeatedly performed until a stop condition is met, at which point the triangle geometry of the current node is queued for rendering. On successive passes over the data structure, the triangles are reassessed to determine whether triangles must be subdivided further, or if neighbors sharing an edge can be merged. When merging, the edge produced by a split operation is removed, reestablishing the parent node's triangle as the current level of detail. Newly merged triangles are queued and the resulting triangles are evaluated for further merging. In the original algorithm, the split and merge operations use dual queues, but other implementations have been proposed that reduce the algorithmic complexity and improve runtime performance by working as a 'split only' instantiation [42]. With a split only approach, the mesh is built each frame starting from the parent node(s). Consequently, the mesh is regenerated each frame; making the algorithm strictly generative instead of incrementally refined. The split and merge operations of the Triangle Bin-Tree are the fundamental routines that give the ROAM algorithm its power.

Unconditionally performing split and merge operations in the evaluation of the Triangle Bin-Tree does not prevent the problem of T-Junctions. T-Junctions are prevented by ensuring that immediate neighboring nodes do not differ by more than one level in the tree. In the situation where a triangle is to be subdivided, the neighbors are

queried to ensure the difference in level of detail will not exceed one. If the split will cause neighbors to diverge, then a forced split of the neighbor is also executed. Forced splits can result in a chain reaction that causes multiple forced splits of many neighbors that share an edge with the remaining two sides of the triangle. For the merging process, merges are prevented when the merge will cause the neighbors to diverge. Managing the triangles in this way is necessary to prevent cracking and to create a seamless surface.

Figure 10 A polygonal split operation.
The longest edge bisection of a right isosceles triangle produces two similar right isosceles triangles. $\Delta ABC \xrightarrow{bisection} (\Delta ABD, \Delta BCD)$

The decision to split and merge triangles is controlled by two conditions. The first consideration is the terrain roughness for a given node and the second is the desired framerate. For the roughness calculation, the removal of vertices in a node is used to precompute the geometric error resulting from the deviation in height. The actual elevation value from the input data source and the elevation of the midpoint of the line that connects two neighbors, as in Geomorphing, is the deviation metric called the roughness. As with other techniques, the length of the displacement is used to determine the distance at which the child node's geometry can be used to achieve the desired visual fidelity. In addition to error-driven level of detail, ROAM offers the option to throttle performance in order to meet a predetermined framerate at the cost of visual fidelity. When processing the tree, the technique can be controlled by submitting triangles higher in the hierarchy, which increases runtime performance but reduces visual fidelity. The

option to trade quality for speed is useful in systems when the framerate lags behind the target framerate by a sizeable margin. Surface deviation and framerate are the two factors that drive the split and merge decisions.

ROAM determines the optimal mesh at runtime by resolving the minimum number of triangles needed to achieve a given fidelity. The optimal minimization of triangle throughput was once considered the primary goal for many algorithms in Computer Graphics; however, it has become more of an antiquated notion in the context of current generation graphics hardware. Initially observed problems with the technique were integrating a texturing scheme and the per-triangle split/merge decision. These early objections were addressed in an unfinished, unpublished revision of the original technique, termed ROAM 2.0 [43]. Regardless of the version of ROAM, they all suffer inherent slowness due to the expensive tree traversal and CPU-bound nature of the algorithm. In addition, most instantiations require excess computational workload to perform per-polygon operations. The per-polygon nature of the original technique precludes it from being suitable for the modern graphics processor. Perhaps, when massively parallel CPU's become prevalent, ROAM will see a rebirth but, of this writing, its original popularity and appeal is diminished.

3.4.4.3 An Improved Block-based Quad Tree

In 1998, Röttger et al. published their work for the *Real-Time Generation of Continuous Level of Detail for Height Fields* [44]. Like the earlier work of Lindstrom et al, the algorithm generates a continuous multi-resolution mesh using a Quad Tree, but differentiates itself by using a top-down approach. Most noteworthy of the technique is

its native support for Geomorphing. Also, the top-down method helps facilitate simplified means for preventing T-Junctions, which had been a problem of the original algorithm.

As with Lindstrom's approach, Röttger makes use of a heightfield whose data is distributed throughout the nodes of a Quad Tree. The root node represents the four outer corners of the heightfield. Subsequent levels additively restore vertices to each quadrant of a parent node, which recursively adds detail back into the mesh by subdividing the geometry with axially-aligned bisections. Each node of the tree is rendered as a single quad, with leaf nodes representing the smallest distribution of vertices possible in the input data source. At runtime, the tree is traversed from the root down towards the leaves. The mesh is rendered using the fewest quads necessary for producing a visually accurate presentation of the terrain to the viewer.

In determining the error metric, the roughness factor for a self-contained subregion of elevation data is precomputed and stored for referential evaluation at runtime. Specifically, the value computed is used at runtime in a comparative operation that specifies if the geometry housed by a node offers enough detail. Given a threshold, the roughness is compared against the split metric. The decision results in the subdivision of the current node and subsequent evaluation of its children or the submission of the current node's geometry to the renderer. A lookup table that identifies the resolution used for each region is maintained. The table is a referential data source that is used to track dependency relations between neighboring nodes. Using the lookup table ensures T-Junctions can be avoided with negligible impact on performance. Many of the principles used in the split decision are derivative of ROAM.

The approach addresses many problems found in the original Block-based Quad Tree method of Lindstrom's et al, but it shares many flaws with ROAM. The tree traversals and random access nature of the technique are suboptimal for system architectures. In addition, it is CPU-bound, which limits its potential for optimality with the GPU and, unfortunately, the problem is inherent in the design.

3.4.4.4 Interleaved Quad Tree

Recognizing weaknesses of previous iterations, Lindstrom would later revisit the level of detail for terrain to propose another solution [5]. The new technique presented a divergence from previous works. It uses a top-down strategy and relies on a specialized, interleaved Quad Tree to address the problems of previous methods. T-Junctions are explicitly handled by the technique and Geomorphing can be included to eliminate popping. The novel trait of the solution is its use of a single data source and the interleaved Quad Tree, which can natively facilitate out-of-core paging using the features offered by the operating system. The method was later evolved into a framework for large terrain visualization [45]. Unfortunately, the overly complex nature of the solution and its reliance on operating system specific system calls make it less attractive as a general approach for terrain level of detail.

At the time Bin-Tree Hierarchies were developed, hardware limitations were such that excess graphics processing would drastically impact overall performance. These methods offered improved performance by submitting a mesh consisting of a reduced number of primitives in order to limit data throughput. They strive to minimize the number of polygons rendered without sacrificing the visual fidelity. To achieve the goal, these techniques construct a tree structure, in which each node corresponds to a level of

detail. The mesh is built by traversing the tree. As each node is visited, the geometry associated to the node is evaluated. Comparative analysis determines which geometry is ready for submission to the renderer and which must be handled further. By rendering a smaller set of geometry, the algorithms achieve improved performance. However, with the recent advances in graphics hardware, Bin-Tree Hierarchies no longer fit the preferred programming paradigm, which puts them at a serious disadvantage. Yet, Bin-Tree Hierarchies are still some of the most widely used and often cited techniques in terrain visualization.

3.4.5 Hierarchical Regions

The third class of multi-resolution techniques for terrain visualization is Hierarchical Regions. With the advent of improved graphics hardware, researchers sought new algorithms that make better use of new hardware features. Hierarchical Regions seek to make better use of the graphics hardware by submitting batches of polygonal data. These algorithms build off of the theory used for Hierarchical Bin-Tree methods, but are augmented to be more appropriate for the hardware. Hierarchical Regions seek to establish a continuous level of detail structure at runtime using a tree based reference structure, like the Triangle Bin-Tree. Central to these techniques is the use of precached geometry that reduces the CPU workload. Precached geometry eliminates costly data transfers at runtime, and allows fast switching between resolution instances.

Although Hierarchical Region techniques share a similar underlying approach, specific algorithms differ in their specifics. Initially, the input terrain mesh is divided into regions. Partitioning into regions may occur on regular or irregular bounds depending on

the underlying data format (e.g. TIN or RGN). Regions are sorted into a tree structure, such as a Triangle Bin-Tree or a Quad Tree. Each region correlates to a single leaf node in the referential tree structure, while the root corresponds to the instanced mesh with the lowest level of detail. Intermediate meshes, from highest to lowest, are assigned to the transitional nodes along the path between the root and the leaf nodes. The process for generating the meshes of different resolution varies from automated methods using an algorithmic simplification process, such as the View Dependent Progressive Mesh edge removal technique, to manual asset creation by an artist. In a given frame, the viewing attributes are parameterized and guide the traversal. As each node is visited, the corresponding instance is assessed to determine whether the geometry of the given node offers enough detail. If the geometry does not offer enough detail, then the children nodes are assessed in a recursive manner. When a subregion meets the criteria for selection, the geometry associated with the node is rendered. Rendering executes without further intervention of the CPU because the geometry data is already pre-cached in video memory. The process is repeated every frame to establish a mesh representation that uses a reduced set of data.

An error metric is computed for each node. The error metric is used for determining when an instance supplies enough visual fidelity. The geometry is uploaded and cached in video memory for quick runtime access. During runtime, the regions are treated as a single object that can be evaluated as a unit. For each region, the error metric serves as the maximum error value for the set. The error allotment is based on the divergence of visual accuracy from the mesh at the next highest level of detail. The region can safely be selected by assessing the relationship of the viewing properties and

the error metric, because the displacement of a single vertex will not exceed the maximum displacement. This guarantees that the rendering stays within a user-defined threshold, which is usually defined in terms of pixels.

In addition to precached geometry, Hierarchical Regions reduce the impact of using a tree. After initialization, each node in the tree is associated to a batch of polygons that form a composite subregion of the terrain at different levels of detail. Representing a batch of polygons with each node reduces the depth of the tree, which contrasts with techniques that correlate a single polygon to a node. The reduced tree depth will speed up traversals and other operations. Algorithms like RUSTiC [29], CABBT [46], BDAM [47], and PBDAM [48] are examples of Hierarchical Regions that demonstrate the use of batched geometry to varying degrees of complexity and success.

In general, Hierarchal Regions can be viewed as hybrid approaches that attempt to achieve better performance by proposing a solution derivative of one or more previously known solutions. Methods of the genre offer improved performance over the Hierarchical Bin-Trees because they do not make excessive transfers of runtime data, which is a result of the pre-cached geometry. In addition, they do not identify the terrain geometry on a per-polygon basis, which reduces the CPU workload. However, these methods are problematic because they can draw in the problems of the methods that they borrow from. The techniques derivative of ROAM suffer from the need to perform a tree traversal, which is CPU limited. Methods that use Irregular Meshes to represent the static geometry set the stage for slowed peripheral operations, such as collision detection. In addition to the problems found in the individual components, hybrid methods are subject to additional problems. T-Junctions and cracking can occur at the edge of discrete regions

and eliminating them can prove non-trivial. Also, precaching the geometry can hinder the incorporation of a Geomorphing solution because it depends on the underlying data representation and the corresponding access patterns. Lastly, these solutions necessarily assume added complexity over their individual counterparts. For Hierarchical Regions, the side effect is that the speedup gained from the inclusion of precached geometry may be offset by the operational and maintenance cost associated with the individual technique.

Hierarchical Regions attempt to resolve problems of Hierarchical Bin-Tree methods by operating on batches of geometry data instead of handling individual polygons. By pre-caching the geometry, these techniques can reduce bandwidth usage and improve computational speed, but they also present a new set of drawbacks. By reformulating Hierarchical Bin-Tree solutions, such as ROAM, these derivative manifestations offer improvements. However, the overheads associated with Hierarchical Regions can subjugate the benefits.

3.4.6 Tiled Blocks

The fourth class of algorithms, Tiled Blocks, is a simple approach that offers excellent performance when executed on current graphics hardware. The modern GPU is a highly parallelized, multi-pipeline processor that can operate on hundreds of millions of triangles per second. Consequently, submitting an excess of polygons to the GPU for processing is a better alternative to burdening the CPU with identifying a minimal, optimal set of triangles. Tiled Block solutions submit geometry in batches that can be quickly rendered instead of trying to assemble an absolutely, perfect set of primitives. A

reasonable excess of geometry is permissible because the processing power of the GPU can handle it, while freeing the CPU to handle other tasks.

Tiled Blocks approaches share much of their philosophy and theory with Hierarchical Regions, in that they both seek to batch polygon data as a means to improve performance. However, even though the motivating factors may be the same, they are two major distinctions that separate them. The first difference is that Tiled Blocks only work with an RGN data source, whereas, Hierarchical Regions allow for TIN data for the regional mesh, as is the case of BDAM [47]. Secondly, whereas Hierarchical Regions make use of a tree for referential decision making, Tiled Blocks do not specify any particular relational structure. Instead of the tree, Tiled Blocks rely on the imposed rectilinear layout of the RGN data source, and partition the world into rectangular blocks. Tiled Block solutions take advantage of the data source's regularity and superimpose the geometric principles that govern the rectilinear distribution of each partition to achieve specializations that are not possible otherwise.

3.4.6.1 Geomipmaps

In 2000, Willem H. de Boer published an online article describing a novel multi-resolution strategy for terrain visualization, called Geomipmapping[49]. At a time when few researchers would diverge from the Hierarchical Bin-Tree methods, de Boer opted to forge a different path, one that reduces CPU overhead by exchanging expensive per-polygon optimizations for grouped polygon processing. Geomipmapping applies simple rules for creating multiple levels of detail for a subregion of the terrain. The resulting instances may use a higher triangle count than could be achieved with other methods, but building and maintaining them requires minimal CPU processing.

The regularity of RGN data offers opportunities for expeditious rendering and data processing during application execution. Elevation data for a heightfield is often stored in a 2D texture map because each texture element (texel) can store the elevation data. The Cartesian coordinate pair used for accessing into a texel are the coordinates on the ground plane, making it possible to perform direct indexing. One important observation of this storage strategy is data compression. In this strategy, it takes one-third of the memory required to store the same number of vertices that it would for an irregular mesh. The more important thing to understand is the implicit correlation of the RGN layout and textures. De Boer observed this relationship, decided to build upon the notion, and devised the technique Geomipmapping. In texture mapping, mipmapping is an algorithm for limiting the cost incurred from texturing a surface at different scales [50]. It limits the computational workload of interpolating a texture's colors when texturing a polygon by using a multiple texture instances. Each instance has dimensions correlative to the polygon's screen occupancy. Mipmapping uses multiple textures of progressively smaller dimensions and quality that are generated by filtering the original, high-detail texture. The filter downsamples the texture, to create the sequence of reduced quality images. At runtime, the hardware is designed to access the most appropriate mipmap in the sequence given the screen-space occupancy of the polygon being textured. In addition to reducing computational workload, mipmaps can be filtered using a digital imaging filter to help reduce runtime visual artifacts like aliasing.

Geomipmapping borrows from the abstract concepts and generation algorithm of texture mipmapping on a terrain mesh described by an RGN data source. In a preprocessing step, a $2^n + 1 \times 2^n + 1, n > 0$ heightfield is downsampled to create a

sequence of $(n-1)$ geomipmaps. A heightfield at level l is the input needed to generate the heightfield data for level $(l+1)$. Using the input dataset level l, every other elevation value is sampled to produce a $2^{n-1}+1 \times 2^{n-1}+1, n>0$ heightfield. In a recursive manner, the newly generated dataset becomes the input dataset for the next, lower level of detail instance. The process is repeated until all geomipmaps are produced (Figure 11). After the geomipmaps are generated they can be uploaded and cached in video memory, to allow for faster rendering of the geometry. At runtime, only one level needs to be processed, since they all offer coverage over the same area; however, the one to choose is driven by the runtime decision making process.

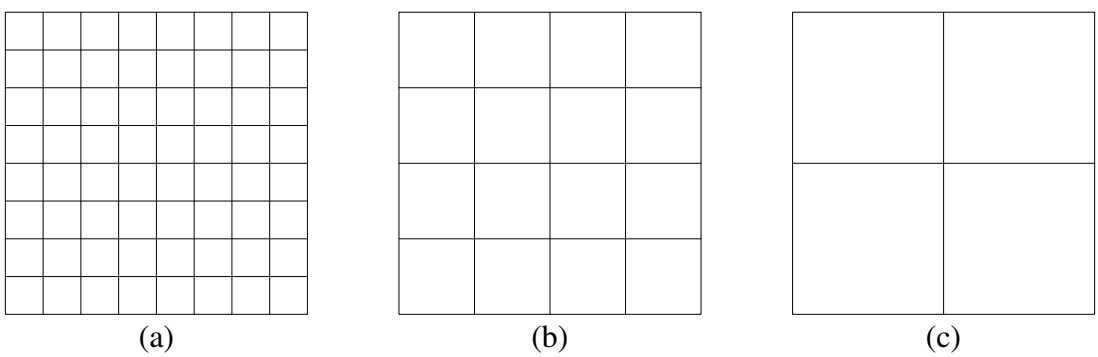


Figure 11 Recursive downsampling a heightfield into Geomipmap instances.
The 9×9 heightfield of elevation data in (a) is downsampled to a 5×5 heightfield and the 5×5 heightfield is used to generate the 3×3 heightfield in (c).

Each instance in the set of geomipmaps provides coverage for a subarea of the terrain. Successive levels cover the same area using half the number of vertices as its immediate predecessor, but with less accuracy. Removing vertices in the preprocessing stage creates divergences between the original topological description of the input mesh and the lesser detailed geomipmaps. The divergence is not noticeable assuming the distance from the observation point and the geometry rendered exceeds a threshold. The threshold is computable based on the displacement that occurs between a vertex present

in a parent geomipmap and the corresponding position it assumes when it is removed. As with other techniques, the relative vertex position is the midpoint between the neighboring vertices of the removed vertex. Using the Pythagorean Theorem, it is possible to compute the length of the vector from the point removed to the midpoint of its neighbors. The computed length is the value necessary for computing the maximum distance at which the vertex can be removed without impacting visual quality, as specified by the parameterized pixel error threshold τ (Equation 3). During the initialization and generation of the geomipmaps, it is necessary to compute the maximum height deviation for all vertices removed from a given geomipmap. Using the maximum height deviation ensures that the distance at which the geomipmap is selected will display a screen-space error that is no greater than the value of τ. During runtime, the current distance to the tile is compared with the value stored in a table of distances. The geomipmap that uses the least geometry and is within the threshold is rendered. The whole process results in the fast display of high fidelity imagery. The reduction in geometry processed is achieved with minimal CPU overhead, making it ideal for use with current hardware architectures.

Equation 3

$$D_n = \delta \cdot C, \text{ where } C = \left(\frac{A}{T}\right), \text{such that}$$
$$A = \frac{nearClipPlane}{\left|nearClipPlane_{top}\right|} \text{ and } T = \frac{2 \cdot pixelErrorThreshold}{viewport_{height}}$$

Geomipmapping is a good solution for achieving multi-resolution display of the terrain. Building off of the theory of texture mipmapping, Geomipmapping extends the idea to geometry. The algorithm is simple and efficient. As with other Tiled Block solutions, Geomipmapping offers improved performance by preprocessing chunks of data

and caching the results in video memory or fast access. However, Geomipmapping can produce visual artifacts in the form of popping and surface discontinuity. In fact, most Tiled Block solutions share in these misbegotten traits, but generalized approaches transgress specific algorithms to solve these problems.

The shortcomings of Tiled Blocks solutions make them an imperfect solution. Tiled Blocks are notoriously susceptible to popping. Although popping is resolvable through Geomorphing, it imbues added complexity and computational workload to the overall solution. Another ill-fated attribute of Tiled Blocks is that they are memory intensive solutions. As a generality, Tiled Block solutions preprocess and store the sequence of tiles simultaneously, which can consume huge amounts of system memory. To further complicate the matter, terrains that exceed available system memory may need to employ a paging solution, which can create direct conflicts with the need to preprocess the terrain tiles.

Another problem with Tiled Block solutions is their propensity to produce discontinuity between neighboring tiles. When such a situation occurs, it is necessary to manage the seams between neighboring tiles. Rendering a block at level l and its neighbor at level k, where $l \neq k$ produces T-Junctions at the shared borders. The T-Junctions often produce visible cracks that detract from the scene, too. It is necessary to address the discontinuity in order to create seamless tiles, which adds further complexity to Tiled Block algorithms.

3.4.6.2 Stitching and Skirts

There are two well known alternatives that serve the purpose for creating visually seamless tiles for Tiled Block solutions: stitching and skirts [1]. Stitching is used to

create absolutely seamless tiles, while skirts are a faster, less accurate resolution. Stitching is often described as an integrated step in many Tiled Block solutions, while skirts can be viewed as an additional step. As such, skirts are defined as Model Services, while stitching is considered an internal step in some level of detail algorithms.

For stitching, neighboring tiles are examined to see if they will be processed at the same level of detail. If they are not, a strip of 'stitching' geometry is generated and processed. In the case of Geomipmapping, it is possible to skip the vertices on the shared edge of the higher level of detail geometry to create a seamless border (Figure 12). The process is repeated for all four sides of the tile that shares an edge with a neighbor and modifications to the geometry set are applied where necessary. The runtime cost for stitching can be high, which is why some systems choose to use skirts.

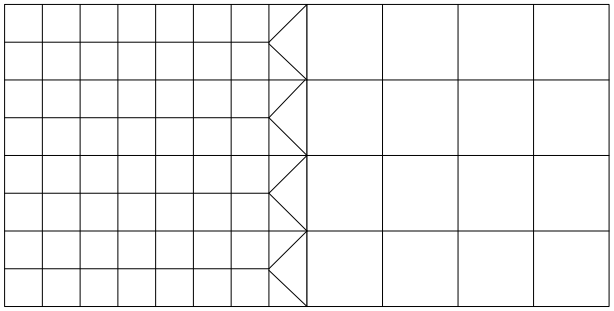


Figure 12 Stitching different resolution tiles.
Cracking is prevented by stitching the higher resolution tile to the edge vertices of its neighboring (lower) resolution tile.

A skirt is a strip of geometry perpendicular to the ground plane and parallel to the tile's edge (Figure 13). Instead of perfectly matching the seams between tiles, a skirt is extended from edge vertices downwards to a position below the lowest point on the entire terrain. The skirt follows the edge around the tile and can mask any discontinuities in the geometry that may be revealed as cracks. Skirts can even be precomputed along with the tile's level of detail at startup, which eliminates any infringement on runtime

performance. As previously stated though, skirts are imperfect. While skirts can mask cracks in the surface, they do not address shading artifacts that can occur at T-Junctions.

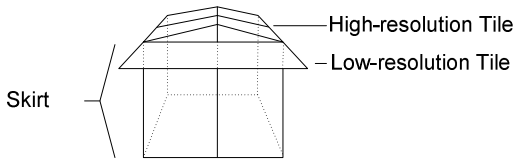


Figure 13 Skirts are used to hide cracking.
The skirt placed around the higher-resolution tile hides the crack that occurs at the edge shared with the low- resolution tile.

Tiled Block solutions are simple algorithms that offer fast and efficient online assembly and rendering of multi-resolution terrain meshes. Solutions, such as [51], have seen a resurgence in interest from researchers because of their suitability for use with modern graphics hardware. The rectilinear layout of each tiled region is easily deconstructed into triangles, triangle fans, or triangle strips for fast processing on the GPU. Batching the geometry allows the GPU to make optimal use of its parallelized architecture, providing excellent throughput to achieve better frame rates and an improved user experience. The simplicity of Tiled Block solutions, coupled with the intrinsic suitability for current hardware architecture, make it a current favorite for use in a number of interactive applications [52]. The primary drawback with Tiled Blocks is the need to manage seams between discrete tiles and necessary integration of Geomorphing, which impose added complexity and impact runtime performance.

3.4.7 Concentric Regions

Concentric Regions are the fifth and final class of algorithms for multi-resolution terrain visualization. As of this writing, the Geoclipmap, in two distinct instantiations, is the only published algorithm that demonstrates concentricity as used for level of detail in

terrain visualization. As with Tiled Blocks, Concentric Regions offer performance improvement achieved by relying on an underlying RGN data source. Concentric regions achieve a speedup by grouping polygons into batches that can be processed in unison, instead of treating each polygon individually. Whereas Tiled Block solutions partition the terrain into discrete, non-overlapping units; Concentric Region techniques work on a focal area using a windowed view of the terrain. The windowed region is centered about the viewer and is composed of multiple levels that overlap. The geometry within the window's boundaries is incrementally updated and processed each frame to define a multi-resolution mesh. The incremental approach eliminates the heavyweight preprocessing task to offer greater flexibility and adaptability to the needs of the system.

3.4.7.1 Geoclipmaps

Losasso et al published the paper *Geometry Clipmaps: Terrain using Nested Regular Grids* as an innovative multi-resolution technique for terrains [4]. As with Geomipmaps, Geoclipmaps operate on batches of polygons to achieve faster throughput and improved runtime performance. Unlike Geomipmaps, they do not subdivide the terrain into a set of 2D linear tiles, but instead regard the terrain as a single region. The key to the approach is that it operates on a user-centric, windowed region of the terrain when processing the geometry. The user centricity offers a highly detailed mesh about the viewer that recedes in fidelity outwards towards the horizon. The Geoclipmap is a fast, efficient, and hardware-friendly multi-resolution strategy for terrain visualization.

Geoclipmaps take advantage of the fact that a heightfield can be stored as a texture map in a 2D image space. The theory behind Geoclipmaps extrapolates the theory of texture clipmaps into the realm of 3D geometry. A texture clipmap is a technique that

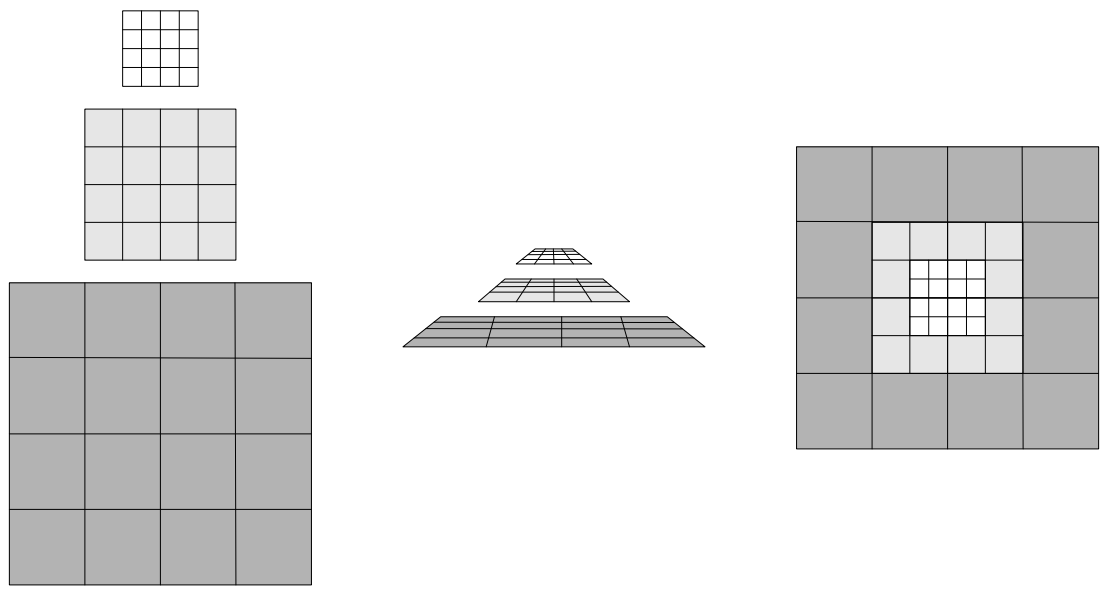
allows a large texture to be used in a scene without requiring that the entire texture be in memory. It accomplishes this by offering a windowed view of texel data that is incrementally updated according to the visible area of the texture. For instance, when only a limited subarea of the texture contributes meaningful detail to the scene, the clipmap algorithm ensures that the visible subimage within the texture is available for use. However, the rest of the image may or may not be in memory at that time. As the view changes and the visible subimage changes, the texture data is updated accordingly. The algorithm updates the memory-resident data to guarantee that necessary texels are presently in memory. The concept of the original texture clipmap algorithm serves as the basis upon which Geoclipmaps are derived with the following key differences:

1. Texture clipmaps require specialized hardware, while Geoclipmaps do not.
2. Geoclipmaps use the distance of the geometry to the viewer to establish the displayed level of detail, whereas texture clipmaps compute per-pixel level of detail using the screen-space projected geometry.

Geoclipmaps attempt to produce a screen-uniform tessellation of the terrain such that every triangle is pixel-sized. The process is generic and ignorant of the surface topology. Techniques for Irregular Meshes and Hierarchical Bin-Trees perform fine-grained, computationally-heavy operations evaluating the mesh geometry to provide per-polygon level of detail adaptation. In an effort to reduce computational cost, methods for Hierarchical Regions and Tiled Blocks can perform less granular operations and, in some cases, are able to offload excess work to the GPU. Concentric Region solutions, on the other hand, use a uniform tessellation of the geometry across the entire terrain to build the surface using each level of detail as a similar, concentric subset of rings. In doing so, the

need for runtime computation of geometry is minimized. It is even possible to abstract the logic even further and offload nearly the entire algorithm to the GPU [53].

The Geoclipmap algorithm is simple and powerful. Performing the technique requires the execution of two distinct phases: initialization and runtime. During initialization, the heightfield is assumed to be a $2^n + 1 \times 2^n + 1$ grid of elevation data stored in a texture map. The multi-resolution instances of the heightfield are generated using a standard mipmap generation technique, exclusive of any filtering [50]. The generated mipmaps are the complete set of the terrain's level of detail instances, and are used during runtime to maintain the active geometry. If desired, it is possible to compress the mipmaps in order to consume less memory. Instead of operating on the entire set of data for each instance, Geoclipmapping operates on a windowed, subset of data each frame. The clipmap region is an $m \times m$ area within a given level of detail instance that is centered about the viewer and meaningful to the scene (Figure 14). At runtime, the view is used to assemble the clipmap region. Startup is the only time that the clipmap region must be completely built, because successive changes will alter an L-shaped subregion of the clipmap as the view changes. During execution, the clipmap regions are stored in a 2D array and uses toroidal indexing to access the elevation data. Toroidal access is necessary to perform 2D queuing operations that allow for incremental updating of the visible elevation data within the clipmap region.



(a) (b) (c)

Figure 14 Geoclipmap regions.
The three overlapping, discrete regions in (a) are projected in (b) and then cropped in (c) to define the multi-resolution concentricity according to the Geoclipmap algorithm.

At runtime, the following steps are performed each frame:

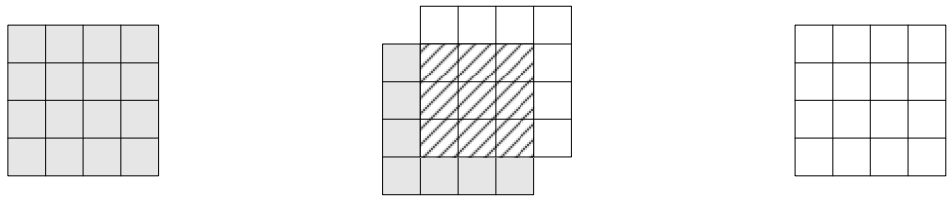
1. *Determine active regions*: Active regions of the clipmap region are determined using the current viewing properties. Each $clip\_region(l+1)$ is a coarser surface representation than its predecessor $clip\_region(l)$. Therefore, the same number of data samples of $clip\_region(l+1)$ offers greater area coverage than $clip\_region(l)$ because the grid spacing is wider. To determine the active regions, a $ng_l \times ng_l$ sample set is derived from each $clip\_region(l)$, where n is the clipmap region size and $g_l = 2^{-l}$. The uniform sizing makes it simple and effective to construct the active regions.

2. *Update clipmap regions*: During runtime, it is not necessary to completely rebuild the active region, because the clipmap regions are incrementally updated. As the viewer moves, it is only necessary to update the clip region in the L-shaped area that has changed (Figure 15). The update comes from the data source, which may be a memory-resident data stored, streamed data store, or procedural synthesis data generator.

3. *Crop active regions*: Active regions may be clipped to speed up processing by choosing the coarsest active regions that offers the desired visual fidelity. Clipping of active regions is performed in coarse-to-fine order, and can be controlled to achieve graceful degradation and throttling. One possible cause for clipping is when the viewer is moving too fast, and the incremental update begins to lag behind. In this case, throttling can be applied to force clipping of detailed geometry, which results in faster processing time. The general case for clipping occurs when the tessellation of geometry is too high and continuing to process it will result in the rendering of overly-detailed geometry. In

either case, the geometry of a coarser level of detail supplants the area under scrutiny to provide the missing surface coverage.

4. Render Scene: Rendering the regions is a straightforward process. For each $active\_region(l)$ the render process submits $render\_region(l)$ to the rendering system, where $render\_region(l) = active\_region(l) - active\_region(l+1)$. Each $render\_region(l)$ is a concentric area of the terrain surface and produces the whole surface as a set of interlocking concentric surfaces.



(a) (b) (c)

Figure 15 Maintaining the correct clipmap region.
As the view changes from (a) to (c), only the L-shaped region of difference in (b) needs to be updated, which is possible using toroidal array indexing.

Each frame the four step process is repeated to achieve high fidelity terrain visualization in real-time. As presented, Geoclipmaps will produce discontinuous surfaces. The discontinuity occurs where the inner edge of a $render\_region(l)$ meets the outer edge of $render\_region(l+1)$. To solve the irregularity at the seam, it is necessary to use stitching or skirts.

Two additional capabilities afforded by the use of Geoclipmaps are compression and synthesis. Compression is possible through the incorporation of a runtime texture decompression scheme. At runtime, the clip regions will need to be updated with new data as the viewer moves across the surface. Texel data is decompressed and extracted from the heightfield being stored as a compressed texture, in order to retrieve necessary elevation values. Through compression, it is possible to retain extremely large terrains in memory. Synthesis allows additional detail to be adaptively added to the terrain at runtime through procedural methods. With synthesis, it is possible to create infinitesimal

detail across an immeasurable area of coverage, without the need to represent the geometry as polygonal mesh data. Synthesis can rely on procedural or functional methods to generate added detail that improve surface fidelity and realism. The added capabilities, compression and synthesis, are two more reasons why Geoclipmaps are viewed as an excellent multi-resolution strategy for terrains visualization.

Geoclipmaps suffers from limitations that make it an imperfect solution. The first limitation is that Geoclipmaps are susceptible to rendering the visible surface at a higher complexity than in other schemes. In the worst case scenario, where uniform tessellation across the entire heightfield occurs, the surface does not benefit from local adaptivity. As such, the overhead of using the solution produces that same output as not using any level of detail technique. The second problem with the algorithm occurs in terrain surfaces with highly irregular surface features, such as tall thin upshots. These features will morph into view late and, depending on the implementation details, the late arrival may create a visible artifact that detracts from the visualization. Lastly, the complexity of the Geoclipmap solution suggests that it may be less appropriate than other solutions for systems that only need to visualize smaller terrains. In particular, Tiled Block solutions may be more appropriate for systems where the terrain dataset and all of the levels of detail can fit into memory. Using Geomipmaps requires an understanding of the caveats in its appropriateness for use.

3.4.7.2 Geoclipmaps (GPU)

The original paper alluded to the fact that the technique could be executed on the GPU [4]. The justification for not doing so in the first place was lack of hardware support for Shader Model 3.0 and, specifically, vertex texture lookups. In 2005, Asirvatham et al.

achieved the task of porting Geoclipmaps to the GPU; thereby, affirming the original authors proclamation [53]. The algorithm itself did not change but, obviously, the implementation was refactored to make more elaborate use of the programmable graphics pipeline. In doing so, the overhead of the algorithm is completely offloaded, completely freeing the CPU. The GPU-based solution epitomizes the ideal, state-of-the-art Computer Graphics algorithm.

Concentric Region solutions are the newest addition to the classes of multi-resolution strategies for terrain visualization. Instead of performing per-polygon operations or partitioning the terrain into polygon sets, Concentric Region techniques use a focal area that reduces in quality as it moves away from the user. The uniform tessellation of geometry along regular grid bounds is used to create downsampled terrain instances. The windowed view is constructed using the distance of the viewer to the region in a radial fashion, which is used to determine when to switch from one level of detail to the next. Using this strategy, the geometry in the immediate vicinity of the viewer is more compact and has higher definition. Areas further away from the center use less geometry, which reduces the amount of processing required to render the terrain Although Concentric Region methods are not without flaw, they are a good alternative for systems that strive to make the best use of current graphics hardware.

Level of detail is the most prevalent area of research in terrain visualization. The need and desire to visualize terrains surfaces that are expansive and detailed justifies the use of a multi-resolution strategy. Level of detail improves runtime performance by reducing the total amount of geometry processed, while maintaining quality that compares to processing the entire mesh. Early techniques sought to optimize the number

of triangles using techniques for Irregular Mesh refinement. Hierarchical Bin-Tree methods perform recursive management of the geometry to build an optimal triangulation using a view-dependent selection mechanism. More recently, graphics hardware has improved, which has lead to the preference for pushing more data through the graphics pipeline and limiting use of the CPU. In response, techniques started shifting focus from optimizing the triangle count to batch processing, such as with methods for Hierarchical Regions, Tiled Blocks, and Concentric Regions. Each algorithm has its own set of strengths and weakness, which are derivative of the intended goals and era under which it was created. Regardless of the specific techniques, level of detail for terrain visualization has generated a wealth of practical knowledge and a variety of useful techniques for use in research and commercial systems.

3.5 Dynamic Terrain

Terrain visualization is used in a number of systems to display topological land features for a variety of purposes. To date, the majority of research in terrain visualization focuses on static terrain and, consequently, many systems rely solely on static terrain solutions. Static terrain accurately simulates rigid terrestrial surface types; leaving the majority of terrain types insufficiently accounted for in virtual systems. Dynamic terrain offers features for terrain representation that make it suitable for simulating non-rigid surfaces. Currently, techniques for dynamic terrain are limited because of the complexity and added overhead. However, as hardware improves and various industries demand more realistic virtual worlds, the inclusion of dynamic terrain will become more and more important.

Unlike static terrain solutions that suffice only to imitate rigid terrain surfaces, dynamic terrain solutions can emulate both rigid and non-rigid surfaces. As such, static terrain is a dynamic terrain subtype. Many strategies used for the visualization of static terrains make use of optimizations and tricks that rely on the rigid nature of the static terrain. For instance, some level of detail techniques are inflexible because they depend on an underlying rigid surface to decimate and restore mesh geometry. When techniques do not translate directly for use in dynamic terrain solutions, new algorithms must be devised to accommodate the needs of a dynamic surface and to supply features lost from the nontransferable technique.

As with many fields of research in Computer Graphics, dynamic terrain research efforts first divergence can occur in the decision of whether to devise a solution that is intended for offline or online rendering. Offline rendering can produce high-quality, realistic terrain dynamics because there are no time critical restrictions placed on the algorithms. The goal in offline rendering algorithms is to achieve the most realistic visualization; however, online rendering is not afforded the same luxury. Online rendering of dynamic terrain requires that the terrain dynamics manifest realistic imagery and operate under real-time constraints. For online methods, the level of realism attained should be reasonably convincing, but does not require absolute accuracy. The realism of the terrain dynamics may be physics-based or appearance-based. Physically accurate models are more realistic, but run slower. On the other hand, appearance-based models trade realism and general use for speed.

Online dynamic terrain solutions are capable of altering the surface to purvey terrestrial information to the observer. Vertex displacement is a primary functional

provision of dynamic meshes and the operational construct that directs that motion is often considered the principle goal of researchers. Many researchers incorporate a Motion Control model that enforces a specific simulation model alongside the dynamic terrain solution. Tightly coupling the motion control to the geometry management scheme is common, but can create intricate dependencies that do not adapt well to other domains. Deformations that result from the displacement of vertices reflect the interaction of the terrain with an object. The realism of the deformation is bound to how well the Motion Control model adheres to the operational constraints and rules imposed by simulation model.

3.5.1 Physics-based Approaches

Physics-based modeling is one approach that can be used to perform terrain dynamics. These methods derive a representation appropriate for use in a visual system by using a knowledge-base that originates from the Natural and Physical Sciences. For highly realistic visualizations, physically accurate models offer unprecedented visuals at the cost of computational complexity and, potentially, runtime performance.

One of the first published works on dynamic terrain attempted to use soil mechanics in order to achieve realistic terrain deformation [54]. The work presents a simplified computational model of soil dynamics and applies it for use in animation and real-time interactive simulations. Specifically, the soil model creates an accurate portrayal of soil manipulation by computing the soil slippage and soil mass displacement. The process simulates erosion of soil as it moves along a failure plane until reaching a state of stability. It is suitable for the simulation and display of terrestrial deformations

that result from actions like pushing, piling, and excavating soil. The physics-based model is suitable for the accurate display of soil displacement.

The algorithm is insufficient and incomplete as a general solution for dynamic terrain visualization. The solution offers soil excavation as the only type of soil manipulation. It does not address soil compression nor does it account for soil composition or moisture content, which would noticeably impact the deformations produced and influence trafficking. The technique's unsupported elements prevent it from being generalized for use in many systems.

Chanclou et. al published a physically-based solution for terrain dynamics that improved upon other works [55]. In the solution, the terrain surface is as an elastic sheet as represented by a particle-mass. The bonds between neighboring particles dictate the possible configurations for the surface. Spatial displacements are driven by interactive forces from external objects. The objects may be either rigid-bodied or soft-bodied. The solution addresses both compaction and erosion using a two step approach. The first step simulates large-scale phenomena, facilitating the simulation and display of soil mass compaction and the displacement. The second step performed is a small-scale refinement that erodes the surface; thereby, facilitating local avalanches and general surface smoothing.

The imagery produced with the method is realistic and, more importantly, the simulation is physically accurate. However, the extensive realism carries a high computational cost that precludes it from being used in real-time interactive systems. In addition, their model does not discuss how to model real-world ground materials, which is necessary if it to be considered a general solution.

3.5.2 Appearance-based Approaches

Appearance-based solutions strive to create a visually plausible rendition of terrain dynamics without the conditional constraint of using a physically accurate model. Instead of rigorous mathematics and complicated mappings, appearance-based solutions use simplifications and fabrications to produce visually convincing phenomena. Appearance-based solutions are not as precise as physically-based models, but in many cases the performance gain justifies the loss accuracy.

In order to convey supplementary information to the observer in animations, Sumner et al. propose an appearance-based solution for the display of dynamic terrains [56]. The solution executes quickly because the computational complexity associated with a physics-based animation is bypassed. It uses a four step execution cycle to create a visually-convincing depiction of terrain surface interactivity. The four steps are:

1. *Collision Detection*: Penetration is detected by casting a ray upwards, from the base position of a column. A collision is detected when the ray intersects an object before reaching the column's maximum height. When a collision occurs, the height at which the collision occurs becomes the new height of the column.

2. *Displacement*: The difference in material is computed using the previous column height and the new height as is determined in the collision phase. The difference mass is either compacted into the local column or is distributed to neighboring columns. The amount compressed is controlled with a user-supplied parameter, the compression ratio (α).

3. *Erosion*: In order to compensate for the displacement step, columns with excess accumulation are eroded to reduce the steep incline. The slope (θ) from a column to its neighbor(s) is computed and, when it exceeds an upper bound, the material is distributed to the lower-contour neighbor. The amount distributed is the average height of the neighboring multiplied by a user-supplied roughness factor (σ).

4. *Particle Generation*: Optionally, a particle generation process fosters the aerial dispersion of surface matter. Matter may adhere to the rigid-body object. The accumulation of matter is dispensed over time into the air as particle masses. When a particle comes to rest on the surface again, the volumetric amount associated with the particle is consumed by the contact region.

Each frame, the four steps are executed and the final surface configuration is rendered to the screen. Continual, incremental updating causes the surface changes to accumulate over time, which gives a tractable history of events and makes the animation more realistic

The algorithm is built with practical application in mind. The authors define five control parameters that will affect the terrain. The surface deformation parameters are intended to ease use of the system by non-engineers. The user-supplied parameters are:

1. Inside slope (θ_{in}) and outside slope(θ_{out}): Controls the shape of mounded terrain material.
    - Small values result in more erosion and a gradual slope.
    - Large values result in less erosion and steeper slopes.

2. Roughness (σ): Controls the smoothness of the surface; contributing to and resulting from ground deformation.

3. Liquidity (θ_{stop}): Controls the amount of erosion per time step; perceived as the wateriness of the terrain material.

4. Compression (α): Controls the amount of material that can be displaced outwards versus downward.
    - $\alpha = 1$: All material is displaced.
    - $0 < \alpha < 1$: Some material is compressed, som material is displaced.
    - $\alpha = 0$: All material is compressed

Although the technique generates convincing results, the approach is not without fault. One shortcoming of the approach is that it can only produce smooth surfaces and particles; it does not support the generation of surface cracks or clumps. Therefore, it is not suitable for clayey terrains that would be subject to such features. A second problem is that there is no mechanism for feedback about the terrain composition, which prevents the integration of any terrain trafficking. In addition, velocity is not taken into consideration during displacement. For certain soil compositions, the piling of soil mass

should be greater in the direction of travel. The last problem is the need to manually adjust rendering parameters to produce a visually-convincing image. The need for manual adjustments suggests that this technique may not be suited for use in an interactive system.

An improved algorithm for the control of terrain deformation was published that uses a specialized data structure, the Height Span Map, to provide dynamic manipulation of terrestrial and granular solid surfaces [57, 58]. The work improves upon previous efforts in the area of dynamic terrain. Specifically, the algorithm provisions for the ability to visualize the displacement of terrain using concave polyhedron and to display displaced granular material on top of an object. The ability to achieve piling on top of objects adds to the realism of the visualization. A simplified form of the algorithm is described by the author(s) in the following three steps:

1. Detect the collision of the object with the surface.
2. Displace the disturbed (granular) surface material.
3. Erode the (granular) surface materials at steep slopes.

The simplified form is a formal declaration of the steps necessary to conduct terrain dynamics. Further refinement of the approach is elucidated as follows:

1. Perform rough collision detection using the bounding geometry of the objects and the terrain surface.
2. Update the Height Span Map for the object.
3. Detect the collision between the object and each column of the Height Span Map.
4. Displace granular material from the columns of the Height Span Map as impacted by external force(s).
5. Erode steep slopes by distributing granular material to neighboring columns.

The algorithm relies heavily on the Height Span Map. The Height Span Map is a 2D matrix of values the represent the minimum and maximum heights of an object. In other words, the objects have a 2D heightfield representation of its extents. The Height Span Map is necessary for the computation of vertex displacements, which is the crux of terrain dynamics.

While the results are realistic, the solution is not without fault. Natively, the technique does not address the issue of scalability. For terrains of large size, the solution will impose a large memory footprint and computational overhead. To combat the problem, the authors suggest the possibility of integrating the solution with a multi-resolution strategy as future work, but they have not done so as of this writing. Also, the surface is assumed to be fairly regular; otherwise, the technique does not perform as well. Another problem with the solution is that it is performed entirely in software. Using the CPU to maintain the Height Span Map and perform all of the collision detection is not good, considering current programming preferences. Finally, the technique may not be practical for use in some systems because the authors only achieve 7-14 frames per second, which is well below the ideal 30 target framerate in interactive real-time systems.

The aforementioned methods for dynamic terrain address the problems faced in trying to accurately simulate the displacement and settlement of earthen materials across a terrain surface. All of the methods have a similar strategy: displace a volumetric amount of the terrain and then erode steep inclines to smooth out the surface, in an effort to remove unrealistic jagged upshots. Some methods are physically accurate, while other use made up parameter sets to control the deformation. The capability of each method is unique, the end results vary, and not one of them is an absolute solution.

3.5.3 Multi-resolution and Dynamism

Of noteworthy concern in the preceding techniques is the lack of focus on integrating dynamics with a multi-resolution method. Most dynamic terrain research efforts focus solely on the development of the simulation model, which controls the deformation. However, they do not address incorporating the deformation strategy into a system that employs a level of detail strategy. In order to successfully use a dynamic terrain solution in large-scale terrain visualizations, the multi-resolution strategy and dynamic terrain solution must be able to co-exist.

3.5.3.1 DEXTER

In contrast to dynamic terrain research seeking to define a simulation model of terrain dynamics, Yefei He's research focused on a specifying a mechanism for dynamically altering the maximal resolution of the terrain mesh within the framework of the system's multi-resolution strategy [30]. Dynamic Extension of Resolution (DEXTER) is a used to add resolution to a mesh in a manner that seamlessly integrates with the level of detail method. Increasing the resolution of the mesh is necessary in cases where the mesh density is too sparse for effectively reflecting terrain deformations; for instance, when an object can exist within the boundary of a single polygon of the terrain mesh. Although the concept behind DEXTER is a general, He's research focused on implementations specifically for Hierarchical Bin-Tree methods. The research was applied to techniques for the purpose of altering the original mesh by incrementally adding more geometry. DEXTER implementations augment multi-resolution techniques, such as ROAM, allowing them to better support mesh deformations. Dynamic Extension

to ROAM is a DEXTER implementation that was integrated into an automotive simulation for improving the visualization aspect of the system [59].

DEXTER allows geometry to be added to an in-memory mesh representation of a heightfield at runtime by forgoing the resolution and density specifications of the original mesh. Hierarchical Bin-Tree methods enforce a rule that prevents splitting the geometry beyond the original limits of the source mesh. The Dynamic Extension to ROAM bypasses this restriction in order to provide dynamic refinement of geometric details. Subdividing geometry is used to add detail to the mesh by performing local updates in the same manner that the Hierarchical Bin-Tree methods execute local tests to identify the geometry to use for mesh in a single frame. In this manner, the approach is used by dynamic terrain systems to add geometry of greater resolution at identified leaf nodes by recursively splitting the polygon(s). The split produces a new set of children nodes in the tree, which uses more polygons to represent the same area. The new, high-detail geometry can be manipulated to create high fidelity terrain deformations. At the same time, the Dynamic Extension to ROAM uses the update process in order to maintain other vertex values, like texture coordinates and material properties. Without DEXTER, a heightfield expands into a full and balanced tree; however, it use can result in a tree that is skewed and asymmetric. Asymmetry occurs because subdivisions can be executed down a single branch of recursively created geometry. A stop condition, such as a maximum depth, is usually employed to avoid the production of overly complex geometry. The Dynamic Extension to ROAM runs at interactive frame rates, which serves as testament to its practicality and usefulness.

While the Dynamic Extension to ROAM is useful, it does not achieve optimal results. For modern graphics hardware, Hierarchical Bin-Tree algorithms are not the preferred approach for terrain mesh representation. Hierarchical Bin-Tree methods suffer from the need to perform local updates and promote per-polygon operations. In addition, the solution is CPU-bound, which further limits its potential for use in modern systems. Another problem with this approach is that the solution presented is only suitable for small, local updates on the terrain, such as tire impressions. Derivative work determined that this strategy can produce larger deformation by using coarse geometry, but even then the deformation is restricted and can only reflect simple structural changes [60, 61]. The Dynamic Extension to ROAM was also extended to offer preservation of vertex properties and relationships with the use of a Direct Acyclic Graph (DAG), which was used to prevent errant changes in the visual presentation of the modified terrain [62]. Large terrain changes would distress runtime performance, possibly preventing the application from running at interactive frame rates. As a concept, DEXTER is a novel finding and useful piece of information that has its place in dynamic terrain visualization; however, the Dynamic Extension to ROAM suffers from shortcomings that make it suboptimal.

3.5.4 General Methods for Mesh Deformation

Although surface deformation and mesh dynamics are not exclusive to terrains, generic methods are not necessarily transferable for use with terrain visualization. For instance, T-DAG is a technique that provides adaptive multi-resolution representation for dynamic meshes with arbitrary deformations [63]. At first glance, T-DAG appears to be a good candidate for use in real-time dynamic terrains on the surface, because it supports

surface deformation and level of detail. However, even though it performs reasonably as a general solution, it is not a good option for terrain visualization because it runs much slower than specialized techniques. In the case of the T-DAG, performance loss can be attributed to the use of a TIN-data layout and slow graph operations, since it has already been pointed out that a TIN data source translates to a slower solution. Another general algorithm for deformable meshes offers multi-resolution potential using a Progressive Mesh strategy [64]; however, its underlying TIN data structure puts it in disfavor for dynamic terrain. While a number of strategies and solutions for dynamic meshes are known, they can not compete with techniques that specifically address dynamic terrains.

Dynamic terrain solutions are necessary to realistically visualize interactive terrain surfaces. They can be used to effectively communicate supplemental information to the observer regarding the terrain composition; thereby, creating a richer and more convincing visual experience. The two approaches for performing terrain dynamics are physics-based and appearance-based. Physically accurate solutions strive to imitate reality by using computational models derived from the Natural and Physical Sciences. Appearance-based models attempt to imitate the visual display in a convincing manner using simple models that are not (necessarily) based on reality. A variety of solutions that address the terrain deformation process share a similar strategy. In general, methods for terrain dynamics perform an initial deformation of the surface and then refine the results to compensate for upshots and peaks that would erode in nature. For a dynamics solution to be truly useful, it must be easily integrated with a multi-resolution strategy for true scalability. Successfully using a dynamic terrain method is a non-trivial feat that will contribute greatly to the realism in a scene.

3.6 Closing

The preceding information presented a comprehensive review of techniques in terrain visualization. The subject matter included a review of topics in spatial partitioning, texturing, level of detail, and dynamics for terrains. Each subject area was further decomposed and specific techniques were examined. It was shown that for each problem, a number of techniques are available that attempt to address the issues. The two goals commonly sought are to improve realism and to improve performance. While both goals are valid for improving the user's experience, the means by which they are accomplished often conflict and compromises must be made. Coordination between solutions to the different problems presents another set of challenges that can further complicate matters. However, when the components are coordinated, the system can reap the benefits by displaying realistic terrain imagery at interactive frame rates.

4. Techniques for Dynamic Terrain

Dynamic terrain is a distinguished approach for terrain visualization that can achieve an unparalleled degree of realism. Many terrain visualization systems are static terrain solutions, which are functionally limited. In contrast, a dynamic solution improves the user experience through increased realism and an interactive environment that more closely mimics the physical world.

Dynamic terrain systems are often more complex than static terrain solutions because they allow for the modification of the surface at runtime. While the inclusion of a deformation strategy alone will suffice as a dynamic terrain solution in many systems, others may require the inclusion of a means to dynamically add geometry for selectively increasing mesh resolution. The approaches for deformation and mesh refinement are the sources of computational overhead associated with dynamic terrain.

In this chapter, we present our contributions to the field of terrain visualization in the areas of runtime mesh refinement and dynamic terrain deformation. The first topic discussed is improving a mesh's resolution for the purpose of dynamism. The idea we evolve is an extrapolation and an elaboration on He's work in dynamic extension to resolution (DEXTER) [30], termed Dynamically Divisible Regions. In addition to presenting some intuitive approaches for polygonal subdivision, we propose our original technical solution for dynamically altering the resolution of a rectilinear grid of elevation points using the GPU. The second half of the chapter is focused on issues regarding terrain deformation and presents our contributions. Our first contribution is an algorithmic specification and formalization of the process for terrain deformation that is

independent of the simulation model. Secondly, we propose an original deformation technique that uses render-to-texture and the programmable graphics pipeline to deform the terrain in real-time. Since the technique adheres to the process specification we identified, it can be implemented to simulate either a Physics-based or an Appearance-based model. The solutions we propose contribute to improving the design and development of a dynamic terrain system that can achieve greater realism in a real-time, interactive visual system.

4.1 Dynamic Extension to Resolution

Terrain meshes are comprised of a finite set of interconnected data points that create a discrete representation of a continuous surface. As with any digital sampling, the analog to digital conversion suffers from lossiness. The problem for terrain elevation sampling is the possibility for excluding local maxima and minima that occur between elevation samples. For RGN meshes, the loss is proportional to the distance between grid points in the network. In order to reduce lossiness, the elevation data sampling must occur more frequently, which increases the dataset size. As the size increases, the need to process and render the dataset impacts runtime performance. Obviously, the conflict of interest between limiting the dataset size and faithfully representing the terrain surface must be overcome.

A fundamental goal when modeling the mesh is to use a sampling interval that limits lossiness, but does not overcompensate and generate an excessively large data set. For static terrain solutions, the generated mesh should be an optimal representation of the terrain with an ideal resolution. Unlike static terrains, the optimal resolution of the mesh may not be absolute for dynamic terrain systems. At runtime, a dynamic terrain is

expected to react to imposing forces by meaningfully displacing vertices; however, a problem occurs if the object imposing the force does not intersect with one or more vertices of the terrain mesh. Under these circumstances, no vertex displacement can occur, indirectly making the deformation strategy inoperative and ineffective.

Systems susceptible to the failure of the deformation strategy may employ a Dynamic Extension to Resolution method to compensate for the shortcoming. Dynamic Extension to Resolution (DEXTER) is a generic term for increasing the resolution of a mesh beyond its original, maximum resolution [30]. The original work on DEXTER only presented implementation details specifically designated for Hierarchical Bin-Trees, like ROAM. Hierarchical Bin-Trees guarantee that the lowest common polygonal shape is the right triangle and emphasize its recursive subdivision nature. These facts were exploited in the Dynamic Extension to ROAM [65], which went against standard protocol by enabling the runtime to recursively subdivide triangles beyond the prefixed resolution of the original input mesh. The technique allows the mesh resolution to be dynamically increased in areas that require greater mesh density, while other areas remain unaffected. The irregular density of the mesh limits additional memory consumption at runtime in an intelligent manner. However, the specific technique is an extension to ROAM, which is a suboptimal continuous level of detail algorithm as pointed out in Section 3.4.4. In general, Hierarchical Bin-Trees are an antiquated approach and, therefore, the Dynamic Extension to ROAM is a misdirected supplement.

4.2 Dynamically Divisible Regions

Rather than focus on a single solution, such as ROAM, Dynamic Divisible Regions (DDR) are a generalized construct for Dynamic Extension to ROAM that can be

specialized to work with a variety of multi-resolution strategies. Specifically, a Dynamic Divisible Region is a subdivision surface specification for terrain visualization. Functionally, a DDR is used to increase the resolution of a region within the terrain mesh and, as a result, it must interweave seamlessly with the chosen level of detail strategy. For RGN meshes, a region is defined along either a right triangle or quadrilateral boundary that contains one or more polygons in its interior. In the case of ROAM and other Hierarchical Bin-Tree methods, a region is analogous to the smallest geometric unit, a single right triangle. For other algorithms, a region may be defined by a group of polygons. In the case of a Tiled Block solution, a region is a single tile represented by a set of $(n+1)\times(n+1)$ rectilinear vertices that form a rectangular area composed of $n\times n$ quadrilateral polygons. In both cases, the Dynamically Divisible Region is a limited area of the terrain, and its regular polygonal shape can be algorithmically subdivided to offer higher resolution.

Dynamically Divisible Regions are formed from sets of polygons that use a subdivision scheme to provide the extension to resolution as specified in DEXTER. The benefit of using Dynamically Divisible Regions is in their potential for parallelizing the subdivision process in two distinct ways. The first parallelization is achievable by subdividing two distinct regions at the same time, as long as there are no intersections between regions. The second possibility is to parallelize the geometry of a region by subdividing the polygons within a single region in parallel, which is well-suited for machine architectures that include multiple CPUs, multi-core CPUs, or a GPU. These two parallelization methods are not exclusive and can be coordinated to create extremely parallel software architectures.

The use of Dynamically Divisible Regions is not without consequence. The issue faced when extending mesh resolution is the potential for excess memory consumption. For Dynamically Divisible Regions, increasing the resolution of a region is done uniformly. In the worst case, where the entire set polygons describing the terrain surface is defined by a single region, the subdivision process will execute across the entire surface and unnecessarily generate an excess of geometry. In the case of extending regions defined by a single polygon, subdividing an individual polygon limits extensional geometry to regions that explicitly request increased resolution. Simultaneously, the performance will be impacted because the process is invoked more frequently and parallelization is minimized. The goal becomes finding an acceptable balance that limits memory consumption and offers a reasonable opportunity for parallelization. However, the acceptable levels for each are largely dependent on application requirements and available system hardware.

4.2.1 Description

Dynamically Divisible Regions are characterized by their ability to be refined from a coarser, lower-resolution mesh into a higher density one. Increasing the resolution of the mesh is useful for two reasons. The first reason to increase mesh resolution is to provide a higher quality visual presentation of the region to the user. It is possible to augment the subdivision scheme with a synthesis strategy that adds fine grain details to improve the visual fidelity of the region. The second reason for refining a region is for the purpose of terrain deformation. The quality of the surface deformation is closely related to the resolution of the mesh being deformed; particularly, in cases where the deformation is intended to simulate visual interaction between two objects. Visually

simulating the influence of an object on a soft terrain surface is exemplary of this situation. As mesh density increases, the deformation can more accurately represent the form of current and residual displacements occurring from interpenetration. As previously mentioned, increasing the mesh resolution leads to more memory consumption and use of processing resources, which can negatively impact performance.

Dynamically Divisible Regions are a generalized construct for offering improved resolution at runtime. In terrain visualization, a terrain mesh is defined by a set of polygons. Initially, the input mesh data describing the surface presents a fixed maximum resolution. Dynamically Divisible Regions allow the runtime system to increase the resolution beyond its initial offering. Resolution is increased by subdividing the regional interior at runtime. Subdividing the region generates a higher resolution mesh that affords greater detail to the viewer and the ability to provide better reflectance of object-surface interaction.

The dimension and polygon count of a region is specified by the system and is closely related to integrated techniques for spatial partitioning and level of detail. In some systems, a region may be the entire terrain, while in others a terrain will be composed of multiple regions. In the case of a terrain decomposed into multiple regions, it may be necessary to incorporate compensatory actions in the level of detail method to prevent cracks between neighboring regions. A region is decomposable into one or more polygons. For each region, the subdivision process is enacted upon all of its polygons. For regions with one polygon, like ROAM, only one subdivision occurs per region, and cracking is natively prevented by the ROAM algorithm. For regions composed of many

polygons, like Geomipmapping, the subdivision process executes many times in a single region and cracking between neighbor regions will necessitate ancillary actions.

A terrain mesh is described by a set of data points that form a continuous surface. The data points are interconnected to create a representational polygon mesh. A region is a subarea of the terrain; therefore, a region is a subset of data points and connectivity information that defines one or more polygons. Since any polygon can be triangulated, a region can be decomposed into the set of triangles that define its surface. In addition, a triangle can be further refined by subdividing it into a set of lesser triangles. Consequently, the resolution of a region can be increased by subdividing its triangulation using a common technique.

Different mesh types encourage the use of specific subdivision strategies, which is practical knowledge when defining the subdivision process for extending mesh resolution. Terrain meshes are modeled as either a Triangulated Irregular Network or a Regular Grid Network. An RGN mesh guarantees the triangles are right triangles, whereas a TIN does not. While in some systems, the primitive polygonal type associated with a region is a triangle, others may prefer to use a rectangle. Both primitive types are valid because they can be subdivided algorithmically to produce self-similar polygons.

4.2.2 Irregular Triangles

A TIN mesh is composed of irregular, or unregulated, triangles that define the surface of the terrain. In order to extend the surface resolution of a region, it is necessary to triangulate the encompassed area. The triangulation of the area is not guaranteed to produce triangles that adhere to any set of rules defining similarity or likeness; therefore, the extension subdivision scheme must handle the most general case. Hence, all triangles

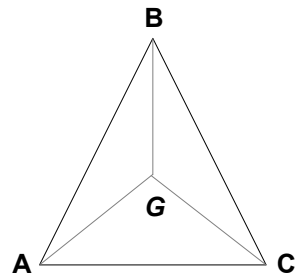
must be treated individually and care must be taken to prevent the introduction of errant discontinuities at neighboring edges. Technique 1 provides one possible subdivision scheme for triangles of a TIN that prevents edge discontinuity and cracks between neighboring triangles. The technique is based on the rudimentary mathematical approach for finding the centroid of a triangle, which achieves the desired results.

Input: Triangle ΔABC

1. Compute centroid G of ΔABC:

$$G_x = \frac{A_x + B_x + C_x}{3}, G_y = \frac{A_y + B_y + C_y}{3}, G_z = \frac{A_z + B_z + C_z}{3}$$

2. Add new ΔABG to triangle list.
3. Add new ΔBCG to triangle list.
4. Add new ΔACG to triangle list.



Technique 1: Subdivision technique for an irregular triangle.

4.2.3 Polygonal Subdivision of a Regular Grid Network

Regular Grid Networks impose greater demands on the subdivision strategy employed for Dynamically Divisible Regions. Subdividing an RGN mesh must produce geometry that maintains the rectilinear structure of the data. The requirement ensures that other methods, such as a multi-resolution strategy, are valid for use with the newly acquired geometry. Fortunately, the data layout is ideally suited for meeting the requirement when subdividing the triangulated mesh, because a rectilinear grid can be decomposed into right triangles or rectangles. Right triangles offer the greatest control over the amount of newly generated geometry, but generate polygons at a ratio of $1:2$. On the other hand, rectangles split at a ratio of $1:4$, which can make them computationally faster, but also offers less control. Lastly, in this discussion a tile is the term used to represent a collection of polygons that can be subdivided at a ratio

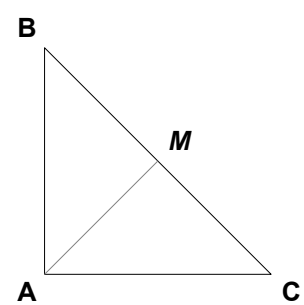
of $1:x, x > 4$. Tiles achieve improved performance at the cost of excess memory consumption as $x \to \infty$.

4.2.3.1 Right Triangles

RGN meshes use a rectilinear grid of elevation data that guarantees the entire mesh can be triangulated into right triangles. Right triangles are subject to a number of special optimizations that can be exploited in the subdivision scheme. One common method for subdividing a right triangle in terrain visualization is the longest edge bisection, which generates two new right triangles for each triangle subdivided (Technique 2). This approach is the fundamental operation of the Binary Triangle Tree used in ROAM [28]. Bisecting a right triangle along the longest-edge produces two self-similar right triangles that are subject to further subdivision using the same procedure. Most often the process is driven recursively and the stop condition is met by reaching a predetermined granularity or depth. A region composed of many right triangles will conduct the procedure multiple times, once for each triangle within its bounds. Longest-edge bisection is the fundamental operation of the ROAM algorithm's Binary Triangle Tree data structure and, consequently, it is used by the Dynamic Extension to ROAM.

Input: Right Triangle ΔABC

1. Compute the midpoint M of the hypotenuse $\overline{BC}$.
$M_x = \frac{B_x + C_x}{2}, M_y = \frac{B_y + C_y}{2}, M_z = \frac{B_z + C_z}{2}$
2. Add new ΔABM to triangle list.
3. Add new ΔAMC to triangle list



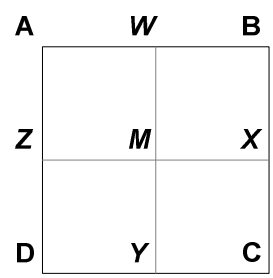
Technique 2: Subdivision technique for a right triangle.

4.2.3.2 Rectangles

In some systems, it is preferred to regard two right triangles in unison as a single rectangular polygon, such as in the Block-based Quad Tree used in [27, 44]. As a result, the subdivision scheme for polygonal interior can be defined in terms of a rectangle. There are a multitude of subdivision algorithms for rectangular polygons that will suffice for the purpose of extension to resolution. One approach would be to subdivide the rectangle into two right triangles that are subsequently subdivided using one of the previously mentioned approaches. Alternatively, a rectangle can be subdivided into a set of self-similar rectangular quadrants (Technique 3). Subdivision of the rectangle using this approach allows for batch processing and polygon generation, which can reduce overall workload by reusing shared computations. The rectangular subdivision approach was used in a Dynamic Extension to the Block-based Quad Tree [30].

Input: Rectangle $ABCD$

1. Compute the midpoint W of $\overline{AB}$.
2. Compute the midpoint X of $\overline{AC}$.
3. Compute the midpoint Y of $\overline{CD}$.
4. Compute the midpoint Z of $\overline{AD}$.
5. Compute the centroid M of $ABCD$.
6. Add new rectangle $AWMZ$ to polygon list.
7. Add new rectangle $BXMW$ to polygon list.
8. Add new rectangle $CYMX$ to polygon list.
9. Add new rectangle $DZMY$ to polygon list.



Technique 3: Subdivision technique for a rectangular polygon.

4.2.3.3 Tiles (Rectilinear Grids)

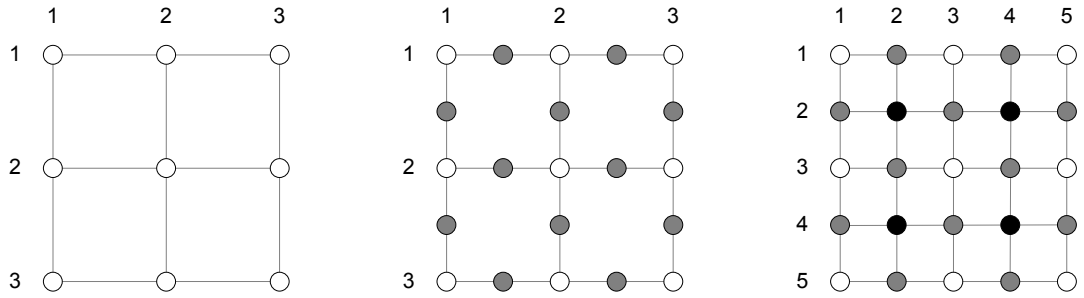
Many terrain systems operate on batches of polygons in their multi-resolution strategy. For the purpose of this discussion, a tile is regarded as a subset of a rectilinear grid data that describes a subregion of the terrain surface. Tiles are defined by a $m \times n$

rectilinear grid of elevation points, where $m \geq 2, n \geq 2$. The elevation set maps to a $m' \times n'$ set of adjacent rectangles, where $m' = (m-1), n' = (n-1)$. Each of these rectangles can be subdivided to produce a new tessellation for its interior area. Subdividing all of the rectangles within a tile generates a higher resolution for tile as a whole. In fact, the subdivision process may be defined as a derivative process that executes the rectangular subdivision scheme multiple times in parallel.

Unlike geometry that can be split independently, tiles are composed of a group of polygons that share edge vertices. As a result, the subdivision process must ensure continuity is preserved. In addition, the tile must be subdivided in a manner that maintains the characteristics of its internal rectilinear structure so that the data is consistent and compatible with the system's concurrently employed algorithms relying on the rectilinear nature of the data. Maintaining the structure and continuity allows the process to be applied recursively without concern for special cases.

Recursive refinement of a tile offers improved performance because a region can be subdivided using fewer operations than independently subdividing the region on a per-polygon basis. In Technique 4, vertices $\{(2,3),(3,2),(3,4),(4,3)\}$ are shared between adjacent tiles and need to be computed and stored once. In this manner, subdividing a tile offers greater performance by optimizing parallelized execution as the dimensions of the tile are increased. However, the trade-off is the potential for producing geometry in excess of what is required. Unnecessary geometry results from the uniform subdivision across a tile's internal area. As either m or n increase, the potential for generating excess geometry increases because the amount of geometry produced is directly proportional to the dimensions of the tile. There is an obvious conflict of interest and,

therefore, it is necessary to find a suitable dimension definition that maximizes performance while limiting the total geometry produced.



Input: Tile (A rectilinear grid of elevation points).

For each set of 4 adjacent vertices in the input tile:
1. Compute the midpoint between horizontally adjacent vertices.
2. Compute the midpoint between vertically adjacent vertices.
3. Compute the centroid of the rectangular region.
4. Add the four newly generated rectangles to the polygon list.

Technique 4: Subdivision technique for a tile.

4.2.4 Hardware Considerations

Technically, for many Dynamically Divisible Regions the subdivision technique can be engineered to execute on the CPU or the GPU. Unlike other runtime terrain system features, such as collision detection, it is understood that the subdivision process is executed with discretion and will execute a limited number of times. Subdivision is performed until a maximum resolution is attained that is suitable for deformation under the impact of all relevant objects in the scene. In fact, the number of subdivisions tends to be relatively low in a single region and is unlikely to occur every frame. Therefore, the subdivision process only has the potential to impede performance on occasion. Deciding which hardware component should handle the subdivision task is a non-negligible task, but the justification for one over the other is not always straightforward.

A terrain mesh is decomposable into a set of geometric primitives that can be subdivided recursively. Dynamically Divisible Regions can exploit this property to extend the resolution of the terrain mesh. For simple geometric primitives, like triangles and rectangles, the computational cost for subdividing a single polygon is low. Comparatively, the computational cost of subdividing a region composed of many polygons is high. In all cases, it possible to devise subdivision algorithms that execute on the CPU or the GPU. All of the algorithms for subdividing geometric entities presented thus far are portable to the GPU; however, not all of them are appropriate. In the case of general triangles and right triangles, the runtime cost in data setup, transfer, and read back is not compensated by porting the single polygon subdivision process to the GPU. On the other hand, subdividing a region composed of multiple polygons, like tiles, may benefit from porting. The decision of which hardware facility to use is dependent on the regional composition, the subdivision technique, and the hardware resource availability.

The generation of geometry resulting from a dynamic extension to resolution method can contribute to improved visual quality when deforming the surface of a dynamic terrain. In many cases, algorithms for dynamically subdividing a DDR that can be executed on the CPU can be finagled to run on the GPU. For simple regions composed of a simple, singular polygon, the CPU is suitable. However, in the case of regions consisting of multiple polygons, like tiles, porting to the GPU offers the advantage of executing on a parallel architecture when generating the new elevation data. In such situations, executing the solution on the GPU may prove to be more expeditious and efficient. Unfortunately, the gains taken from porting subdivision from the CPU to the GPU are limited due to the nature of the problem. Since the dynamic process is not called

consistently and, in practice, is executed a limited number of times, the impact of these techniques to affect performance is restricted. Even still, porting to the GPU will contribute towards reducing CPU workload when the subdivision process is called, which promotes greater consistency and stability for the runtime.

4.2.5 Subdividing Tiles on the GPU

Tiles are well-suited for subdivision using the GPU. A tile is a subset of rectilinear elevation data that represents a subarea of the total terrain mesh. The locally maintained geometry is subject to subdivision in situations that require a dynamic extension to resolution. The subdivision process must produce a tessellation of higher resolution that maintains the rectilinear structure. The single CPU version of the algorithm described by Technique 4 accomplishes regional subdivision through a multiphase approach that computes the midpoints for the horizontal neighbors, calculates midpoints for the vertical neighbors, and determines the centroid for each interior rectangle. The sequential execution of a single processing pipeline impedes performance, because these computations could be executed in parallel. The parallel processing architecture of the modern GPU can be used to exploit this option, which can benefit runtime performance.

Specialization of the subdivision process for tiles is necessary, in order to take advantage of the GPU. The following algorithm describes how the GPU can be used to execute the subdivision process for converting a $m \times n$ tile to a higher resolution $m' \times n'$ tile.

1. *Encode the $m \times n$ tile of elevation values as a $m \times n$ texture*. Elevation data stored in a regular grid can be mapped from 3D space to 2D image space through straightforward encoding. Assuming y is the elevation sample, each vertex (x, y, z) in 3D space, can be encoded by normalizing the y value, if

needed, and storing it in the pixel located at (x, z) in the image plane. The end result is a $m \times n$ texture suitable for use by the GPU.

2. *Load the texture into video memory*. With the elevation data in a suitable format, the $m \times n$ texture must be physically transferred to the video memory of graphics hardware.

3. *Render the texture to a screen- spaced quadrilateral, with a $m' \times n'$ viewport*. Prior to rendering, the viewport and rendering properties must be configured appropriately. In order to change the resolution from $m \times n$ to $m' \times n'$, the viewport is defined to be $m' \times n'$ and the perspective set up as orthographic. Once the configuration is set, a screen-spaced quad that reaches to the extents of the viewport is drawn using the encoded elevation data as its texture map. When the quad is drawn the fragment processor computes the interpolated values for missing texture coordinates using a linear image filter or through a fragment program. Upon completion, the render target stores the sample set that represents the region at a $m' \times n'$ resolution, albeit encoded.

4. *Read the $m' \times n'$ render target back into main memory*. With the newly computed elevation values ready, the encoded $m' \times n'$ texture is read back into system memory by accessing the render target.

5. *Decode the $m' \times n'$ texture into a $m' \times n'$ tile of elevation values*. Through an inverse transformation, the $m' \times n'$ texture is decoded into a $m' \times n'$ set of elevation data. In the case where y is the pixel value, each pixel in image space located at (x, z) can be decoded by rescaling the y value and storing it as a unique (x, y, z) value in the vertex data set.

The process presented performs the same duties and offers the same functionality as the algorithm described in Technique 4; however, the entire process is executed using the GPU. Figure16 shows an aerial view of an input mesh at a fixed resolution. Figure 17 shows the same mesh after it has been retessellated to a higher resolution using Technique 4 on the GPU. Unlike the CPU approach, the GPU method can be used to resample the elevation data for the purpose of increasing or decreasing resolution, without further specialization. The key in sampling via the GPU is the transformation of data from 3D space to 2D image space. In image space, the problem is equivalent to stretching a texture, which the hardware is readily capable to do without intervention.

The inverse transformation restores the encoded 2D image data back into its 3D counterpart, except the region's resolution is changed. The primary bottleneck in the process is the transference of data from system memory to video memory and back. If the latency of transfer time is outweighed by the performance gain from porting, the method is optimal. The gain in performance is proportional to the number of polygons contained by a region, because of the potential for exploiting the parallelism of the GPU. In addition, the latency times should decrease as graphics hardware improves, which makes the GPU port a stronger candidate in the long term.

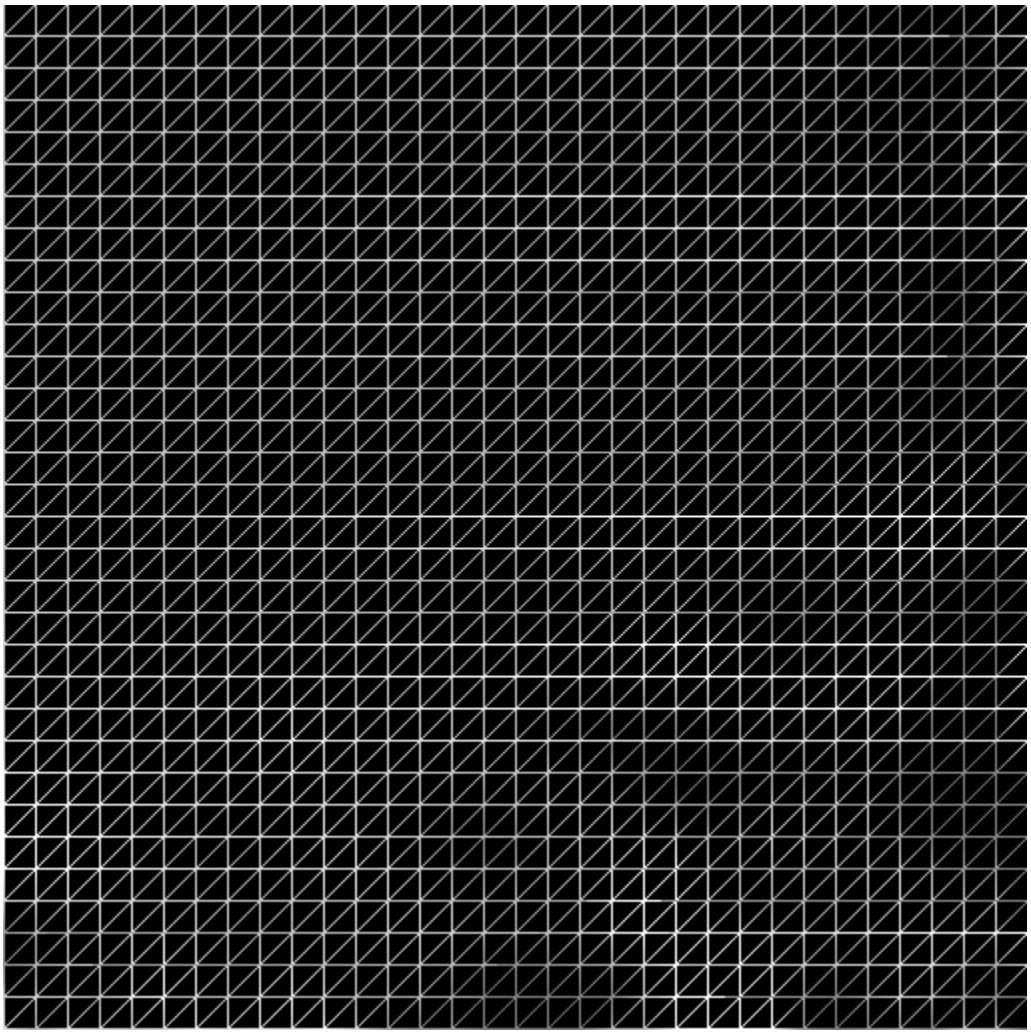


Figure 16 The input mesh.
The original input $33m \times 33m$ terrain with $1.0m$ posts.

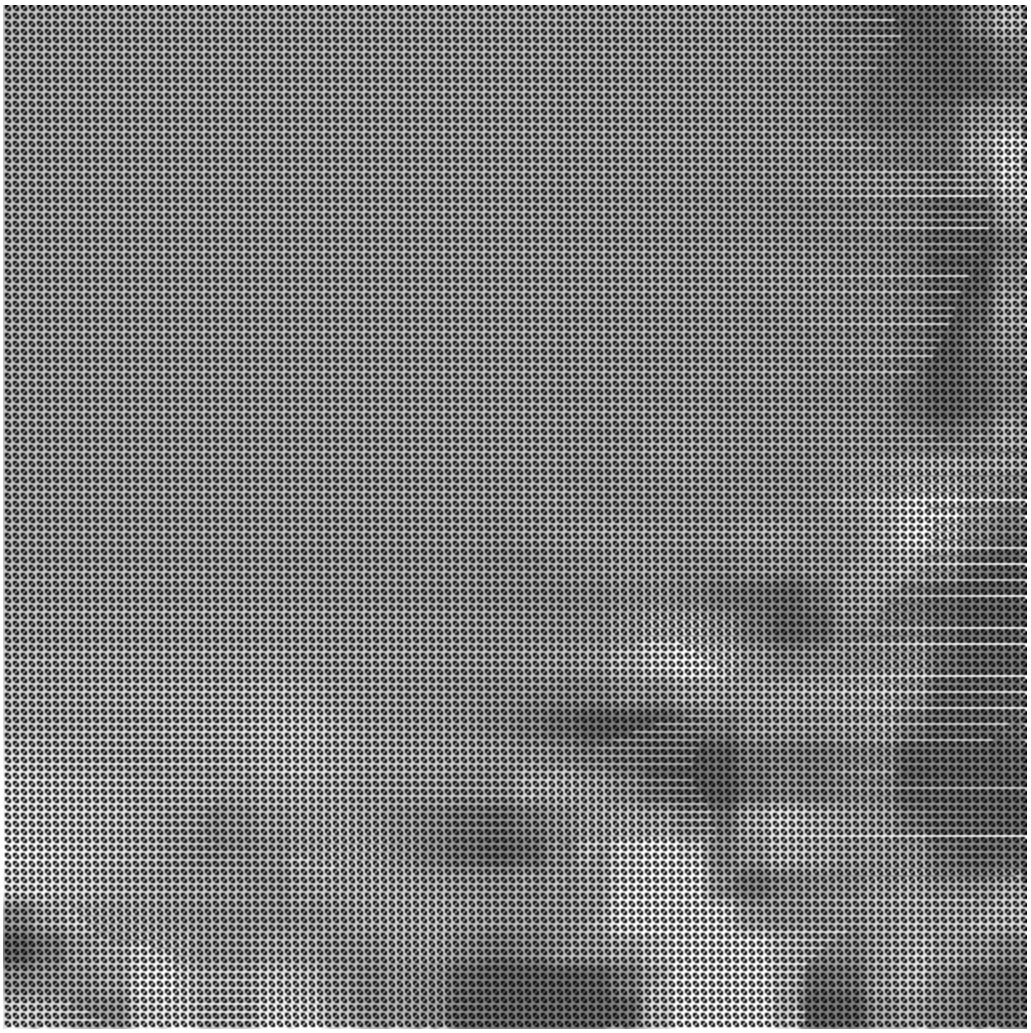


Figure 17 The dynamically divided mesh.
The $33m \times 33m$ terrain after the dynamic subdivision process with $0.25m$ posts.

Dynamically Divisible Regions are a specialization of subdivision surfaces for terrain meshes that offer runtime extension to mesh resolution. Terrain meshes can be decomposed into subregions defined by one or more polygons that form a continuous surface. Each region that is a Dynamically Divisible Region can be extended to provide an increase to the resolution for its area of coverage. A Dynamically Divisible Region is a practical solution for terrain systems that require runtime compensatory actions for improving the mesh resolution. Regionally-contained polygons are subdivided in accordance with an algorithm that generates additional data points to achieve higher

detail. Since all polygonal meshes can be triangulated, the challenge becomes selecting the best subdivision technique to use and which hardware should undertake the task.

4.3 Terrain Dynamism

Dynamic terrain is most notably characterized by an interactive, deformable surface. In order to modify the surface, it is necessary to incorporate a runtime deformation strategy within the terrain visualization solution. For many topics in Computer Graphics surface deformation can be defined in terms of a mathematical function; however, in the case of terrain visualization, surface deformation is a reactive response to the simulated force imposed by other objects in the scene. The majority of research on deformation strategies in dynamic terrain visualization focuses on the underlying simulation model that computes the redistribution of displacements occurring after initial vertex movement. Physics-based approaches, like [54] and [55], direct the movement by applying physically-inspired rules to create a convincing virtualization of surface interactivity. Appearance-based methods, such as [56] and [58], use invented parameterization sets that simplify calculations at the cost of computational accuracy, but produce a reasonably convincing simulation. However, for our purposes, the rules for distributing the vertices based on a simulation model do not qualify as a complete deformation strategy.

4.3.1 Deformation and Rendering

Ideally, terrain deformation would be defined according to the laws of Soil Mechanics, allowing both the simulation and visualization systems to imitate the real world perfectly. Unfortunately, current computational power prevents the ideal from becoming a reality. Therefore, partial simulations models or fabricated algorithms are

adopted to provide reasonably convincing imitations. Different simulation systems are tolerant to different levels of complexity and exactness in the visualization of terrain changes. Although they both work towards a similar goal, physics-based methods are more computationally correct but slower, while appearance-based methods can achieve visually-acceptable, inaccurate results at a much lower computational cost. Both approaches share the same goal, but with different emphases. In response, this research originates a loosely-specified algorithmic approach that supports either driving factor: computational correctness or performance. The technique is flexible enough to allow for various instantiations to be specified and to be developed independently, but also offers enough commonality that comparative analysis is possible. The technique can be described by the following four stages:

1. Determine the initial displacements for terrain vertices, as moved by the imposing object during the current frame.
2. Redistribute the displacement offsets in accordance with the soil simulation model.
3. Update the periphery vertex attribute data affected by the deformation.
4. Process and render the (deformed) terrain mesh.

The first stage attempts to compute the incrementally displaced vertex position as it is changed by forces currently imposed. The second stage is the analogous to steps in physics-based and appearance-based methods, which attempt to define a compensatory process that imitates the visual, if not physical, characteristics of Soil Mechanics. The third stage is an intermediate step of the deformation process that is used to ensure peripheral data associated with vertices remain consistent with the recently altered

topography. Lastly, the terrain mesh is processed and rendered to screen, which presents the deformed terrain to the observer.

4.3.2 Overview

The four phase approach for dynamic terrain is a complete solution for the specification and development of deformation techniques in dynamic terrain. Although specified in general terms, each phase explicitly identifies the necessary tasks for deforming the terrain surface.

In essence, the first two steps of the technique are the simulation system that is used to deform the terrain. Computing the initial vertex offsets and the subsequent redistribution is functionally descriptive enough for identify deformation routines of varying complexity, but is done without instilling constraints or restrictions about the technique. For instance, both the physics-based method found in [55] and the appearance-based method presented by [56] employ a multi-step process that accounts for displacement and redistribution. It is conceivable that any deformation model can be defined in terms of these two actions. As such, it becomes possible to identify compatibilities and differences between methods through the generalization.

Generalizing the deformation process distinguishes it from the implementation. Deformation techniques that are overly complex or offer poor performance will discourage the use of a dynamic terrain solution. The computational cost incurred is determined by the algorithmic design and implementation of the deformation strategy. In addition, there are a wide variety of possible solutions under this specification; ranging from highly-complex physics simulation to quick-and-dirty approximations. Unlike a specific technique, the general strategy affords flexibility that can be exercised when

evaluating what's acceptable in terms of speed and computational accuracy. Consequently, it is possible to defer the decision regarding precision to meet the needs of the system without inadvertently discouraging its inclusion.

4.3.3 Displacement Computation

Terrain modeling produces a terrain mesh composed of an initial configuration of vertices, edge-connectivity data, and vertex attributes. The position of the vertex is the most noticeable component of the terrain's surface description. When deforming the surface, the vertex position is altered to make known interaction between the terrain and any objects imposing force on the surface at contact points. In order to simulate interaction between a terrain and an object, it is necessary to offset vertices contacted by the object. Specifically, offsets for each affected vertex must be computed to eliminate any visual anomalies where an object interpenetrates the terrain when a deformation should occur. Offset values are assigned in accordance with the Motion Control model to prevent errant vertex displacement. For instance, heightfield-based systems will employ a constrained motion control agent that limits the displacement to the elevation only. Limiting changes to vertical offsets prevents movement of vertices in the direction of the ground plane, which serves to preserve the rectilinear structure of the data source. Intentionally, the initial offset of the vertex simulates the compression of the terrain. For many soft surfaces, compression is the most prevalent alteration resulting from interactivity.

4.3.4 Displacement Redistribution

Although simulating the compression exclusively is an option, redistributing the displacement values is preferable for simulating many soil compositions that are subject

to local avalanches. Local avalanches occur when the shears stress from a steep incline overcomes the shear strength and the soil travels down the failure plane until it settles in a stable state. Clayey soils have a high cohesion factor that can overcome the inclination of the soil to crumble in on its self; however, many other soil types can not. After compression, the ridgeline of loose, granular soil types around the impact area will experience localized avalanches that remain active until stabilization between the shear strength and shear stress is reached. At the same time, on the interior of the impact area, some soil compositions may partially decompress once the compaction force is removed. In order to simulate surface dynamics in addition to compression, it is necessary to enact a second simulation step that processes the surface using a soil simulation model. The secondary activity affords the simulation an opportunity to refine the topographical distribution of displacement values for the purpose of accurately reflecting the simulated soil composition. For a number of systems, the redistribution process is the main contributor to the visual system for implicitly conveying the soil composition and makeup.

4.3.5 Vertex Attribute Maintenance

Once the topography for a frame is established, it may be necessary to perform maintenance on vertex attributes and periphery data. Often, vertex data has locally associated information that contributes to the final surface appearance in the rendered image. There is potential for the pronouncement of visual artifacts when these attributes are not properly updated against the current surface configuration. Vertices are fixed in static terrain systems and, therefore, the data associated to a vertex does not necessitate dynamic runtime maintenance. Deforming a surface allows vertices to change during

program execution and may, therefore, require additional processing to coordinate the data associated to a vertex as it changes. Consequently, additional processing resources must be allocated to ensure that periphery data coincides with the surface as it changes during program execution.

Examples of relevant vertex attributes include normals, texture coordinates, and colors. Deforming the surface may result in the need to recompute vertex normals, for the purpose of lighting calculations. Texture coordinates may need to be modified to mitigate the appearance of visual artifacts on revised polygons. Vertex colors can be modified in creative ways that offer visual cues or compensations to an observer. Logically, any data associated to displaced vertices may require maintenance routines to coordinate the local changes with the overarching surface deformation.

4.3.6 Final Processing and Render

As a final step, it is necessary to process the terrain data and render the image to the output device. During this phase, the runtime implementations for Modeling, Rendering, and Animation techniques, as required by the application, are subject to invocation. Spatial partitioning will limit the total set of geometry submitted for processing. Level of detail will further refine the geometry, while Geomorphing compensates for popping. The application of textures, shading, and atmosphere are used to improve the detail of the terrain. Out of this process, the final image of the interactive, deformable terrain is generated and sent to the output device.

The aforementioned process provides the high level overview of what is executed during the simulation and visualization of a dynamic terrain solution. Each frame, the four-stage process is executed by following the same sequence: compute the compression

displacements, redistribute the displacements, update the vertex attributes, and process-and-render the terrain dataset. In this manner, the terrain is managed each frame and terrain dynamism is achieved in real-time.

4.4 Terrain Dynamism on the GPU

In an effort to improve the field, it is necessary to evolve dynamic terrain by developing better algorithms that limit the overhead associated with dynamism. The primary source of overhead stems from computing the initial offsets and redistributing the displacements. These tasks are central to dynamic terrain visualization, but the computational complexity incurred in the implementation deters the overall attractiveness of such solutions. Reducing the assumed overhead by offloading additional work onto the GPU is a good recourse that alleviates the burden and improves the appeal of a dynamic terrain solution.

When an object imposes force onto the surface of a terrain, the terrain exerts a force of its own in the opposite direction. If the force exerted by the terrain succumbs to the force exerted on the terrain, the topology will change. It is the job of the Application Logic in the main application system to implement the simulation models for objects in the scene. As such, the Application Logic component bares the burden of calculating the orientation of its associated objects with respect to the terrain. In this manner, the simulation model of an object can be coupled to the object, independent of the terrain system. After the simulation model has determined the orientation and position of the objects in the scene, one or more objects may interpenetrate the terrain surface. Even though the simulation model is running correctly, the visual model will suffer from lessened realism because it does not accurately reflect surface interactivity. In order to

remedy this situation, it is necessary to deform the terrain in a convincing manner where the visual presentation accurately imitates real world terrain interaction.

Surface deformation occurs when an object in the scene imposes a force on the terrain that necessitates the simulation of terrestrial responsiveness. The overview for dynamic runtime deformation provides a high level guideline in the development of a terrain deformation technique. A number of techniques for terrain deformation are possible; however, not all are ideally suited for the task. Many proposed solutions are incomplete because they concentrate solely on the redistribution process. In addition, the known solutions do not attempt to make explicit use of graphics hardware. The following presents a novel technique for realistic terrain deformation that makes excellent use of the GPU as the means for reducing computational overhead incurred from using the technique.

4.4.1 Displacement Computation

The first task in deforming a terrain surface is the computation of the initial offsets occurring from soil compression. In this solution, compression occurs as a reactive measure wherever an object penetrates the surface. The computed offset is a value associated to a vertex that, when applied, displaces the vertex position to a location that eliminates the object-terrain surface interpenetration. In most cases, the presence of an object will invoke displacements for multiple vertices each frame. Calculating the offset is a per-vertex operation that has the potential to greatly impact performance. Nonetheless, through the application of compression to each vertex, the surface appears to conform under the simulated force of the object.

In order to alleviate the CPU from an excessive workload, the computation of compression offsets can be offloaded to the GPU. In order to use the GPU, the elevation data must be in a format useable by the GPU. The rectilinear layout of elevation data of an RGN mesh is suitable for use by the GPU, but to be accessible the application must prepare the render state and uploading the data to video memory.

The first step for uploading the data is to configure the rendering state. The configuration of the scene is as follows:

1. Configure the render target as a depth-buffer render target.
2. Configure the rendering system to use an orthographic projection.
3. Configure the viewport to a $m \times n$ area of the render target.
4. Translate the viewing position to the center of the $m \times n$ region where the object causing interpenetration is located.
5. Lower the eye point below the lowest elevation point of the region.
6. Rotate the viewing direction to be perpendicular to the ground plane.

Although the preparation of the render state is used by a number of subsequent steps, the perspective view and application's camera state must be preserved for the final processing phase. Specifying the render state and configuring the camera in this manner encapsulates the region of interest in the viewing volume of an orthographic projection (Figure 18). This setup allows the object and the terrain to be rendered from underneath the terrain, looking upwards, while the depth buffer specification is used for computing the compression displacement values. The result of preparation is a windowed view of the terrain centered about the area of interest that makes data transformation and operation on the GPU possible.

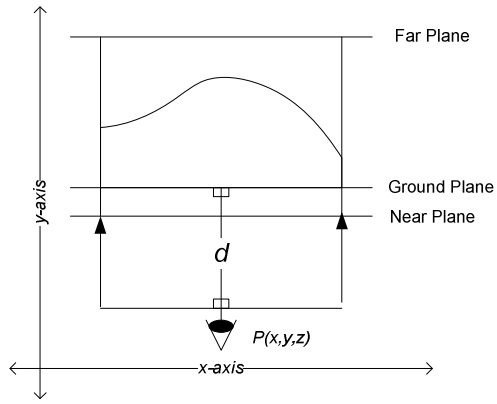


Figure 18 Camera state for deforming the terrain on the GPU.

After configuring the camera, the elevation data must be transformed from the system memory format to one that the GPU can use. A transformation process can be used to encode the elevation data of a rectilinear region into a $m \times n$ texture called a Dynamically Displaced Height Map (DDHM). The transformation process is used to encode the data and transmits it from system memory to texture memory. If the range of elevation data in the mesh region is not $[0,1]$, then the values will need to be normalized. Inversely, a scaling factor can be applied to a normalized data set in the vertex processor to decode the true elevation value of the vertices. The vertex (x, y, z) in 3D space is encoded into a 2D image space by mapping the elevation value into the (x', z') pixel location, where $(x' = \frac{x}{m}, z' = \frac{z}{n})$ (Figure 19). Once encoded, the elevation data is uploaded from system memory into texture memory, where it is accessible by the graphics processor. The transformation process only needs to be executed when the encoded texture for a region is not resident in texture memory. All subsequent processes that access the elevation data can be specified read-only, which will preserve validity for future iterations of the algorithm. Upon completion of the transference, the region's

geometric description is appropriate for use by the GPU, both in terms of format and physical storage medium.

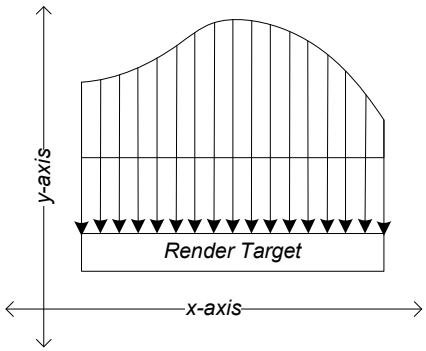


Figure 19 Mapping elevation data into the render target.

After the render state is configured appropriately and the geometry for region-of-interest is available in texture memory, it is possible to compute the initial offsets induced by surface compression using the GPU. The first step in computing the compression offsets requires the terrain data be decoded from the texture and stored in the render target that is replicating functionality of the depth buffer. By rendering the elevation data decoded from the texture, the render target retains the region's elevation data from the perspective of below the surface. Next the object or objects that are within the bounds of the region are processed and the depth buffer functionality afforded by the render target is used to determine surface visibility. In this manner, the object geometry that is closer to the camera than the terrain elevation will be handled by the fragment processor (Figure 20). With the camera below the lowest elevation point of the terrain, only the vertices correlated to object geometry that is penetrating the terrain surface are processed. In addition, each fragment processed in the render target corresponds to exactly one vertex in the $m \times n$ region. The depth buffer is used to compute the compression offset for each vertex that is being compressed by an object. As each fragment in the render target is

processed, the actual compression offset is computed by the fragment program. The displacement value is stored into a second $m \times n$ render target called the Offset Map. Each pixel in the Offset Map has a one-to-one mapping to the vertices in the $m \times n$ DDHM, which also corresponds to the $m \times n$ dimensions of the input region. The generation of the Offset Map uses the parallelized fragment processor to quickly and efficiently calculate the compression offset of each vertex. The parallel computation of vertex offsets contrasts the slower CPU-based method that relies on sequential vertex processing. With this approach, the offsets are expeditiously calculated on the GPU, leaving the CPU free to handle other processing tasks.

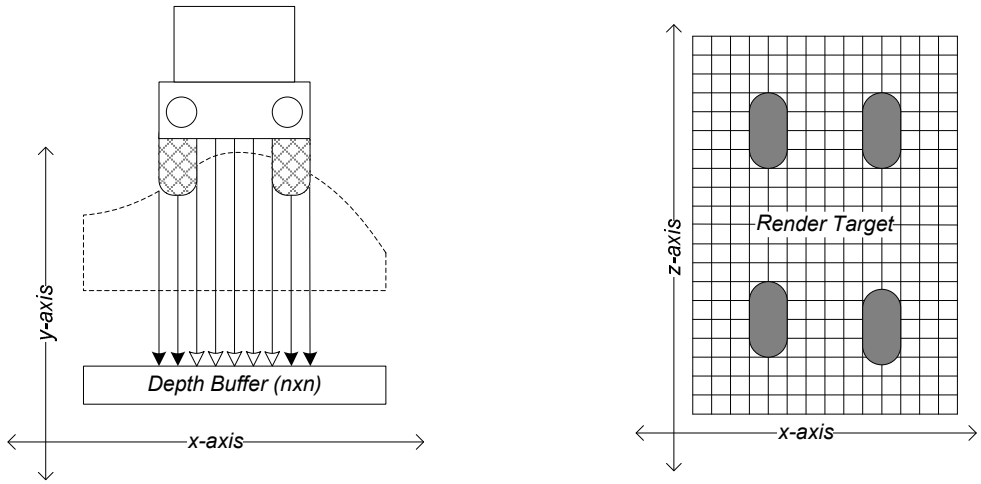


Figure 20 Recording displacement values into the Offset Map.
Darkened arrow heads correspond to visible object surfaces that are recorded into the Offset Map. Hollowed arrow heads are discarded since the terrain mesh occludes the object and no penetration or compression is present.

4.4.2 Displacement Redistribution

Upon completion of the first task, the Offset Map stores normalized compression offset values for vertices displaced under the compaction of an object. The Offset Map is a persistent record of object penetration imposed on the regional surface over time. The deformation recorded in the first step is a mirrored-impression of the object, which is

ideal for highly cohesive soil compositions that retain form, like clay and mud. For other soil types, including granular soils, it is necessary to perform a compensatory redistributive process because the stability of the soil would not retain the compacted impression.

As with computing the offsets, it is possible to offload the redistribution process onto the GPU; once again, alleviating the CPU from assuming the additional computational workload. In order to make use of the GPU, the redistribution process requires that the Offset Map be located in texture memory. In the preceding step, the Offset Map for a $m \times n$ region was generated by operating on a $m \times n$ render target. If the preceding steps use a render-to-texture, then the Offset Map is already present in texture memory. In addition, the same camera configuration, render state, and the vertex list are reusable for the purpose of the redistribution step. As a result, redistribution carries minimal overhead and integrates seamlessly with the first stage as presented.

Redistributing the offset values on the GPU is a straightforward task because the preparatory work for the process is already done. The strategy is to render the Offset Map into a second $m \times n$ render target and, while rendering, the fragment processor is used to resample the offsets. Initially, the Offset Map is maintained as a 2D image in texture memory. Using the render state and mesh vertices for the region being deformed, the Offset Map is drawn to a $m \times n$ render target. In some cases, it may be preferable to render the object and not the region in order to limit the amount of processing undertaken. In both cases, each fragment in the render target maps to exactly one vertex in the DDHM and the Offset Map. The initial displacement value for each vertex is read from the Offset Map, which shares a common (u, v) coordinate with the DDHM. In

addition, samples can be read from neighboring values to compute a less rigid formation through the application of inventive filters. The sequence of lookups and blending is particular to the implementation and can range from highly complex to obviously simple. Physics-based soil simulations can use the fragment processor to compute the shear strengths and shear stresses in order to determine the physically-correct distribution of soil across the area. Appearance-based solutions can use simple weighted averages or other image filters to blend the surface, thereby lessening the rigidity produced by the compression calculation. In either case, the $m \times n$ render target output from the filtering process supersedes the input Offset Map and becomes the current Offset Map for the region.

The redistribution of compression offsets allows the surface to react in a dynamic manner that affords greater visual realism. In addition to the clay-like soils that can be simulated otherwise, filtering makes it possible to simulate granular soils compositions. The redistributive process described makes use of the GPU in a simple and straightforward manner. The technique presented is analogous to the practice of image filtering using the GPU. Rather than focus on a specific process, the proposed strategy is presented in a generic manner. The intent is to dissuade the reader from thinking in terms of physics-based and appearance-based simulation solutions, because these are implementation dependent issues. With this technique, it is possible to devise customized implementations that meet the needs of the system, without imposing restrictions that would otherwise limit it from supporting the full spectrum of soil compositions and simulative complexities.

4.4.3 Vertex Attribute Maintenance

The geometric surface of a mesh is insufficient for realistic terrain imaging. In addition to positional data, vertices will carry associated attributes that contribute to the realism of the final image. For most terrain meshes, attribute data is defined during the offline modeling phase or it can be computed using the mesh for referential purposes during initialization. For static terrain solutions, once the information is known, it remains fixed and can be cached for reuse during execution. Unlike static terrains, dynamic terrains are not stagnant, which necessitates proactive maintenance and coordination of the vertex attributes as the surface is deformed.

Dynamic terrain is characterized by a deformable, interactive mesh. As the topography of the terrain is deformed, vertex positions are changed and the data associated with the vertices may require runtime services, as well. For example, the displacement of a set of vertices may invalidate the associated normals, texture coordinates, and colors. The use of cached values would negatively impact the rendered image, because the associated information is no longer valid given the current surface formation. Incorrect normals can result in the display of unrealistic shading artifacts, because shadows and lighting are erroneously computed. Invalid texture coordinates can produce exaggerations in texture stretching in areas where vertex displacements have expanded the shared polygonal surface area beyond an implicit threshold. Inappropriate vertex color assignments may result in visual discrepancies that conflict with the intended display of qualitative information. The displacement of vertices impacts the potential of peripheral data because the very intent of this data is to compliment the polygonal mesh. Obviously, as the polygonal mesh is modified, the relevant data may require modification

too; therefore, specialized maintenance routines that coordinate the vertex attributes with the geometry may be necessary to preserve image quality at runtime.

Many of the common vertex attributes can be computed during initialization and these methods can be adapted to work during runtime, as well. Both normals and texture coordinates can be mathematically computed in a variety of ways. Computing the normals of rectilinear grid of vertices is rudimentary, if not obvious. The designation of texture coordinates for the various texturing strategies is also straightforward. Vertex colors can be manipulated through lookup tables or functional processes according to system-specific needs. During runtime, these methods can be executed using the newly displaced vertices to determine the currently correct values. However, computation should be limited to the areas that have undergone deformation during the frame. Limiting the area reduces the overall workload, which contributes to an optimal performance.

Vertex attributes include the entire set of satellite data associated with a vertex that are used to improve the visual realism of the rendered mesh. Although normals, texture coordinates, and vertex colors are the most common, each application is free to designate and use unique attributes for specialized purposes. Since the vertex attributes employed in a system are application-specific, it is neither reasonable nor in the scope of this research initiative to present specifics algorithms and techniques for addressing such concerns. In lieu of proposing a specific technique for updating vertex attributes, this research simply acknowledges that the potential exists.

For each vertex attribute in the system, it is important to evaluate the impact resulting from surface deformation and to execute the proper course of action in the case

where visual discrepancies arise. Although runtime maintenance of vertex attribute data imposes additional workload, its absence can result in visual anomalies that affect the system's ability to produce realistic imagery.

4.4.4 Data Process and Image Render

Processing the dataset and rendering the scene is the final stage of the solution that gathers all the information and displays the final image. The task at hand is to assemble the various datasets into set of collective information that is useable by the graphics pipeline. Data must be organized and submitted to the renderer, which processes the conjoined information to generate the imagery. The output image portrays the deformed terrain mesh that appears to have interacted with the object; thereby, simulating the reactivity and dynamism of physical terrain.

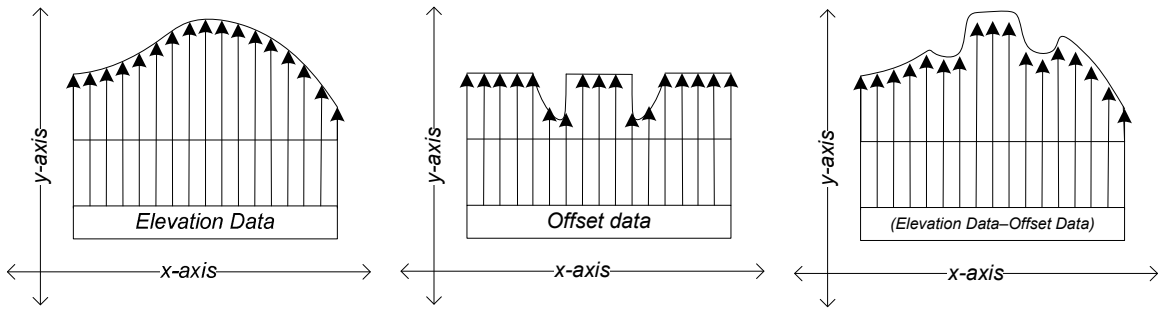
At this stage, the system must restore the rendering state, gather the relevant data, submit the data to processing pipeline, and execute the processing pipeline. Unlike the previous phases that operate on focused areas of the terrain where objects interact with the terrain, the final stage must process all of the terrain within the application view's frustum and render it to the output device. Therefore, the camera configuration is restored to use the values of the application's camera. In addition, the projection is reverted to a perspective projection, which restores the viewing properties and three-dimensional aspect to the scene. In the simplest system, the entire terrain mesh is submitted to the renderer; however, this is suboptimal and can unnecessarily overburden the processing pipeline. For advanced systems, the terrain solution will employ a spatial partitioning algorithm to cull large batches of geometry and a level of detail algorithm to further

reduce the total polygon count. In all systems the geometry is collected according to the internal rules of the application and submitted to the graphics processing pipeline.

The graphics processing pipeline is designed for the exploitative use of the GPU. Many tasks that were executed on the CPU with a fixed-function pipeline can be offloaded to the GPU and the programmable pipeline. The elevation data is encoded in the DDHM and displacements offsets are stored in the dynamically updated Offset Maps for visible regions. Both the DDHM and the Offset Map are encoded as textures and stored in texture memory that is readily accessible by the graphics processor. The availability of the vertex data in texture memory is important because Shader Model 3.0 supports vertex texture lookups, which lets the vertex processor access the information. Using the vertex textures lookup functionality, the DDHM and the Offset Map are used as displacement maps. Each vertex uses a common texture coordinate to access both the elevation height and its displacement offset from the textures. First the elevation data is decoded from the DDHM (Figure 21a). Next the offset corresponding to the elevation is retrieved from the Offset Map (Figure 21b). Finally, the two values are blended to establish the final position of the vertex (Figure 21c). The computed difference is the height at which each vertex is located according to the persisted deformation history. The process is performed on each vertex to create a faithfully-deformed, interactive surface.

In addition to the amalgamation of displacement maps, supportive operations and algorithms may be executed in the vertex and fragment processors to contribute to the final image. Within the vertex processor, Geomorphing can be employed in an effort to minimize popping that can result from the use of level of detail. Texture mapping,

shading, and atmospheric effect techniques can be incorporated into the fragment processor to alter the presentation of the terrain, making the scene more realistic.



(a) (b) (c)

Figure 21 Data is combined in the vertex processor.
The rendering process uses the encoded elevation data in (a) and offsets in (b) as displacement maps to generate the final, deform terrain surface (c).

Rendering is the final set of processing handled each frame to generate the imagery on the output device. The resulting image is a dynamically altered surface that displays object interactivity with deformation persistence. In addition to the final assemblage of the rendered topography, a number of supplementary tasks are handled at this time. Although not necessarily a part of the render process itself, spatial partitioning and level of detail techniques may be executed in the determinacy of the geometry to be processed. The polygons submitted are the items that will undergo further graphics processing to define the final image produced by the graphics pipeline. For deformation purposes, the DDHM and the Offset Map are blended within the vertex processor and the computed value is used to displace the vertex. During vertex and fragment processing methods for Geomorphing, texturing, shading, and atmospheric effects are also realized.

4.5 Large Terrains and Dynamism

Many research initiatives in terrain visualization are focused on resolving issues in handling large-scale terrains. The most common problems associated with large

terrains are memory consumption and processing workload; both of which can be exorbitant. Level of detail is the most common solution in terrain visualization, used for the purpose of handling large-scale terrains. The need to coordinate the dynamism routine with the multi-resolution strategy may be complex, but not impossible.

The proposed technique for terrain deformation is compatible with systems that support large terrains through the integration with level of detail. The only stipulation regarding compatibility is that the level of detail technique must operate on a terrain along rectilinear bounds (i.e. a Regular Grid Network mesh). The rectilinear alignment of the data is necessary for mapping the data from system memory into texture, via the DDHM encoding. In addition, the regular alignment can be used to partition the terrain into distinct regions where each region is a discrete subpart of the larger terrain. The concept is not new to very-large scale terrain solutions that can incorporate a paging technique. Notably, each region of the terrain can maintain a DDHM and Offset Map for its interior area, which is necessary for deforming each region individually. By iteratively applying the dynamic terrain technique to each region, it is possible to deform the large terrain as a whole using this divide-and-conquer approach.

The definition of the regions used for deforming a large terrain may be fixed or dynamic, which can make for two unique approaches in handling the terrain deformation process. The boundaries defining a fixed region are specified once and remain valid for the duration of application execution, while dynamic regions allow for the re-specification of boundaries during program execution. Although both approaches are equally valid, the use of one over the other may be more intuitive given the level of detail technique used by the terrain system. For instance, a Tiled Block algorithm would be best

integrated with a fixed region approach, while a Concentric Region approach would more easily integrate with a dynamic approach.

The differences between a fixed approach and a dynamic approach for the purpose of deformation are simple. A system using fixed regions for deformation may require multiple passes in a single frame, while dynamic deformation region will require runtime maintenance. In a fixed system it is likely that an object will intermittently cross regional boundaries (Figure 22). When this occurs, the deformation strategy will need to be executed once for reach region the object occupies. In the case of a scene with a single object whose bounding area is less than a single region, the algorithm will have to execute, at most, four times in a single frame. On the other hand, a dynamic region uses the object as a referential construct and dynamically redefines the region boundaries to be centered about the object (Figure 23). In this manner, the region always encapsulates the object, which eliminates the need to execute multiple passes of the strategy for one object. With both approaches, as more objects are added to the scene, the offset computation, redistribution, attribute maintenance, and rendering tasks will be executed more times per frame; however, this is an unavoidable consequence. The benefit of using a regional approach is that it constrains the computational workload assumed by limiting the maximum area processed for deformation in a given frame. Large terrains introduce a number of unique challenges to terrain visualization. Many of the hardships incurred are remedied through the inclusion of level of detail. In order to provide dynamism with large terrains, the deformation routine can be incorporated to work using regional terrain partitions, instead of on the entire terrain. The two possible regional approaches are fixed and dynamic. For both approaches, the details of an implementation are dependent on the

algorithms that the deformation is to be integrated with. Yet, independent of the details, both succeed at providing the facilities for dynamic terrain functionality in large terrain systems.

(a) (b)

Figure 22 Deforming a terrain using fixed regions.
A spherical object is entirely housed within a single fixed region (a). After moving, the object occupies four distinct regions (b). The deformation routine is performed once for each region that the object holds occupancy in during a frame of execution.

(a) (b)

Figure 23 Deforming a terrain using a dynamic region.
A spherical object is housed within a dynamic region (a). After moving, the object remains within the bounds of the dynamic region. A runtime maintenance routine updates the boundary definition of the region as the object moves.

4.6 Closing

Dynamic terrain systems are an improved terrain visualization solution because it can actively portray an interactive surface. The commonplace presence of static terrain systems has established a number of basic features that are expected of the average terrain visualization solution. In addition to these features, dynamic terrains solutions must provide functionality to demonstrate its interactive nature. While the inclusion of a dynamic extension to resolution procedure is optional, the presence of a deformation technique is mandatory. Dynamic extension to resolution methods can dynamically

increase mesh density to offer higher quality deformation. Dynamically Divisible Regions are a generalized specification that can be used to work within the constraints of most systems. It has also been shown that it is possible to implement at least one DDR instantiation using the GPU. Whereas extension to resolution is optional, the deformation process provides the means for interactively altering the surface, which is the defining characteristic of a dynamic terrain. While the deformation process can be specified in terms of a compression displacement and redistribution of vertex positions, additional processing is also employed to coordinate periphery data and render the final terrain. Section 4.2.2 presents an algorithm that makes explicit use of the GPU to deform terrain. The output imagery visually simulates interactivity and dynamism that improves realism. The two components, dynamic extension to resolution and the dynamism strategy, create the distinction between a static and dynamic solution. When employed correctly, the result is an enhanced visual experience and, possibly, a more precise simulation.

5. Dynamic Terrain System

5.1 Introduction

The thesis presents a number of novel ideas and technical solutions for application in terrain visualization. To support the work, we have developed a terrain subsystem implementation that uses these ideas to create a dynamic terrain solution for an off-road ground vehicle simulation system. The terrain solution is combined with an automotive simulation system in development by members of the Hypermedia & Visualization Lab (HVL) within the Department of Computer Science at Georgia State University. The application demonstrates the utilitarian nature of a dynamic terrain solution as it pertains to visually-enhancing ground vehicle simulations. It is shown that the integration of an interactive deformable terrain improves the observable realism, which contributes to an increase in the visual credibility of the system.

5.2 Goals

The purpose of a terrain visualization system is to provide the user with a realistic presentation of the landscape that includes information to the scene while conveying form, texture, and presence. In order to provide a truly realistic presentation, the terrain must supply the observer with a convincing presentation of the surface. For many systems, the surface is assumed to achieve a reasonable level of realism through the use of texture mapping, surface shading, and atmospheric effects. Although the rendering components can improve visual realism, they are insufficient for visualizing dynamism and interactivity. The omission of dynamic deformation limits the degree of realism that can be achieved in terrain systems that are intended to characterize soft, loose soil

compositions. The best remedy for the misaligned use of a static terrain system is to replace it with a dynamic solution.

For the purpose of ground-vehicle simulation in off-road conditions, the terrain is most suitably represented with a dynamic terrain system. Unlike roadways, off-road driving conditions are considered suboptimal. For ground-vehicles, the suboptimal nature of off-road conditions implies the presence of non-rigid surfaces. The lack of rigidity means that the vehicle will cut and carve into the terrain as it traverses the surface. Consequently, the terrain should provide the capability to display the remnants artifacts left behind from vehicular contact through the presence of tread marks. Taking into account the possibilities for faithfully visualizing off-road driving, a dynamic terrain visualization system is the best approach.

In addition to dynamism, the solution must incorporate a number of techniques commonly used by solutions for terrain visualization. Among the features to be included are texture mapping, shading, and level of detail. These features contribute to the overall visual appeal of the terrain and ensure that performance is maintained. The application acts as a testament to the practicality of augmenting a terrain solution to include dynamism. The final product seamlessly integrates hybridized algorithms that work cohesively to form a complete solution for the simulation and visualization of ground-vehicles in off-road conditions.

5.3 System Design

The application is comprised of different pieces that have been constructed across two separate, but related, research initiatives. The first part is the vehicle simulation component and the second is the terrain system. The two pieces were integrated to

produce an application suitable for the simulation and visualization of ground vehicles that can interact and influence a dynamic terrain surface. While both pieces contribute equally to the final product, the two systems were developed independent of one another. Consequently, the terrain system was created as an autonomous unit to be consumed and used by the simulation system. Therefore, we limit the discussion of design and implementation issues to the terrain system and do not attempt to describe the vehicle simulation whatsoever.

The terrain system was developed using the design architecture of the component framework as a guide. A number of simple, yet effective, techniques are implemented for each component to achieve improved visual quality and to increase performance. Figure 24 is a (simplified) class diagram that describes the terrain design. The class diagram divulges the interface of the terrain and reveals a number of methods of the system that contribute to the final solution. Class attributes are unique to each object. Included in the solution are attributes such as vertex position, texture coordinates, and normals that are necessary for any terrain system, regardless of it being TIN or RGN, single or multi-resolution, and static or dynamic. In addition, it supports dynamism in conjunction with level of detail, which makes it truly unique.

The terrain solution can be characterized according to its qualities. First, the terrain only supports the regularly spaced set of elevation data offered by an RGN data source for its mesh representation. An RGN mesh was decided upon because the regularity and rectilinear structure of the geometry makes it suitable for specific algorithmic simplifications that serve to improve performance and enhance the visual display. Also, according to [41], RGN mesh processing can be faster than TIN mesh

processing on current graphics hardware. The solution also uses a custom level of detail method derived from [49] and [1], distinguishing it as a Tiled Block solution. While both Tiled Block and Concentric Region approaches are equally appropriate for use with modern graphics hardware, Tiled Block solutions require less complex runtime maintenance and, for our purposes, is the more appropriate solution. Lastly, it is as a dynamic terrain solution because it incorporates the GPU-based method for terrain deformation. Integrating the deformation technique exhibits the functional use of terrain dynamism while incurring only a nominal performance penalty. The terrain solution incorporates a number of features common to terrain visualization systems in conjunction with the deformation technique, making it suitable for exhibiting the functional and practical nature of a dynamic terrain on current computer hardware.

5.4 Component Design

The solution was developed using the component framework for terrain visualization. For each component, one or more techniques were specified and implemented into the solution to fulfill the requirements and meet the needs of the system. The framework is well-suited for discussing algorithmic decisions at a high-level, because it affords others the opportunity to analyze and compare the design decisions in relation to their own. Comparatively analyzing systems in the context of the component-based architecture is more meaningful because it provides a formalized common infrastructure, as opposed to a comparison based on randomly chosen system features.

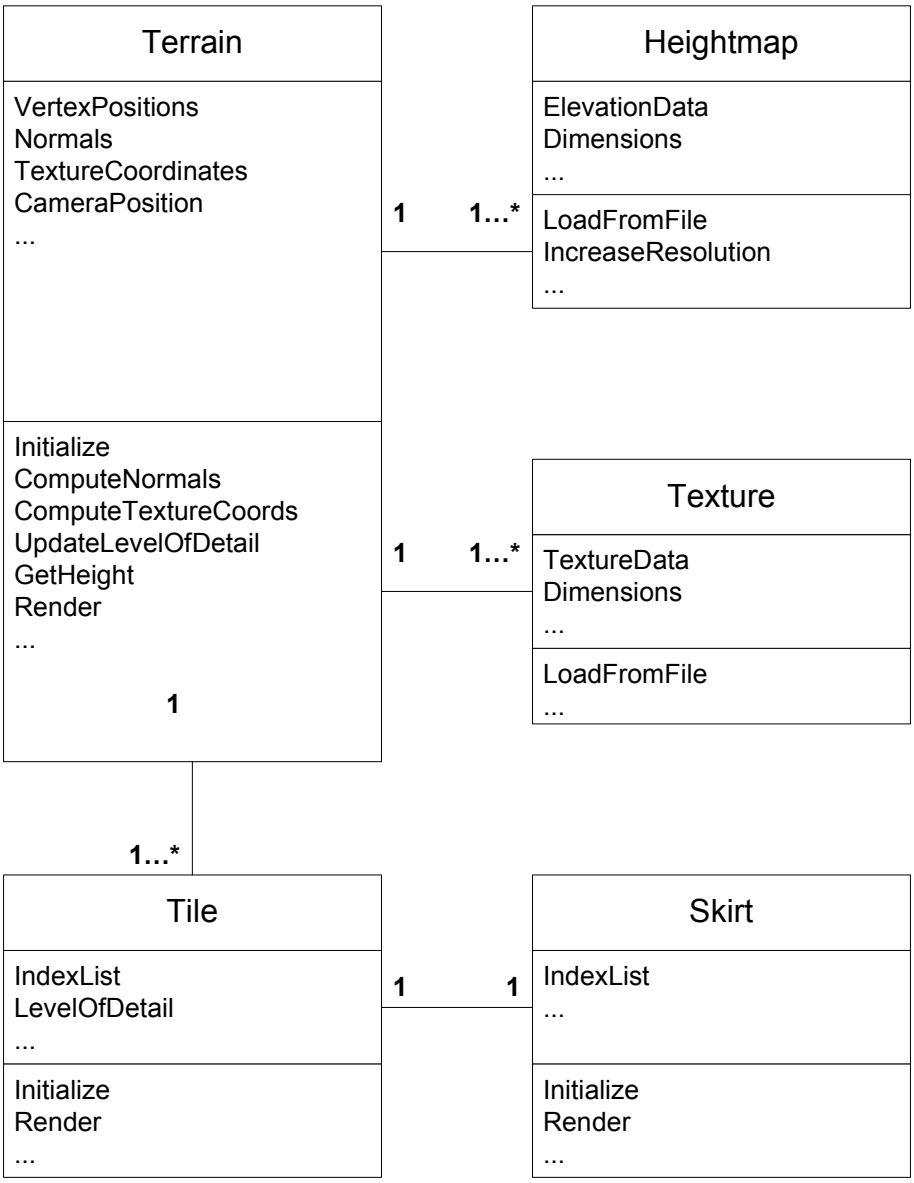


Figure 24 Class diagram for the terrain system.

5.4.1 Modeling Components

The terrain mesh is derived from a set of elevation data stored in an offline height map. The elevation values define the geometric structure of the terrain displayed to the user and interacted with by virtual objects. The elevation set is stored as a $n \times n$ heightfield in 8-bit raw image format that is, subsequently, loaded by the application during startup. The raw image is transformed into a set of three dimensional vertices

using the transformation function $f(x, z) = y \rightarrow (x, y, z)$. Transforming the entire dataset produces the three-dimensional rectilinear mesh in memory. After the mesh is assembled, vertex attributes, including normals and texture coordinates, are dynamically computed by the terrain object. In addition to loading the elevation data, the initialization process will upload all relevant textures from offline storage and move the image data into texture memory.

During initialization, the resolution of the mesh is updated using the tiled subdivision technique for Dynamically Divisible Regions presented in Section 4.1. Resolution is improved to ensure that the vehicle(s) in the system are capable of spatially influencing the topology during program execution. In the system, the entire terrain is treated as a single region, because the vehicles are free to travel anywhere. In the case of this system, runtime tessellation was not a requirement and pursuing that path would have added unnecessary complexity. As a consequence, the DDR is uniformly subdivided across its entire surface once at startup. Specifically, for this system, the scene is well-defined because both the required maximum resolution and terrain extents are known at startup. Therefore, it is possible to conduct the subdivision process once during initialization. In this case, the potential for excess in total polygon count is acceptable in comparison to the complexity and workload associated with performing the subdivision process at runtime.

5.4.1.1 Spatial Partitioning

The mesh is composed of a rectilinear grid of elevation points, which makes it well-suited for spatial partitioning using tiles. Therefore, the mesh is partitioned into rectangular regions using its native rectilinear arrangement and neighboring tiles share

vertices along their common edge (Figure 25). Although each tile is self-contained, it is important that changes to edge vertices are shared between the neighboring partitions.

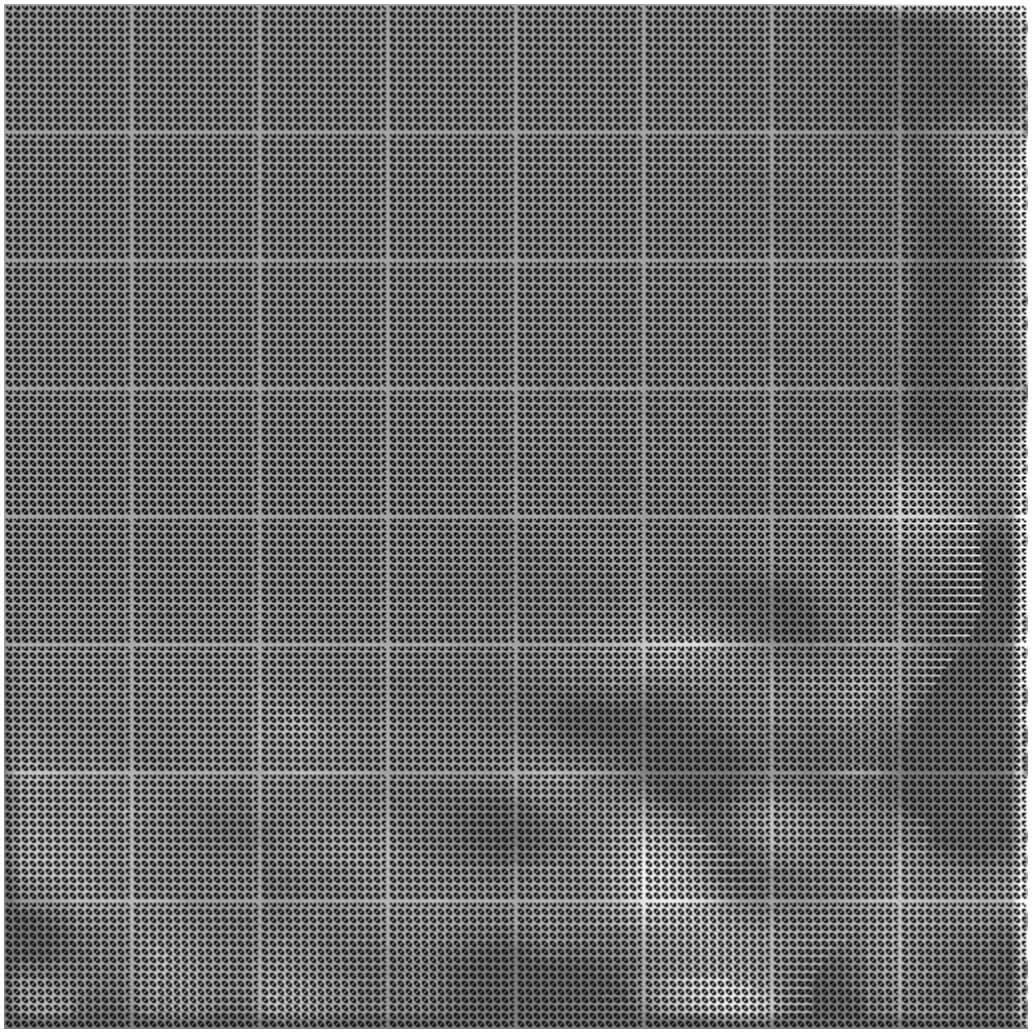


Figure 25 Orthographic view of the spatially-partitioned terrain mesh.
The 256×256 terrain is partitioned into related, but independent, 32×32 tiles.

Terrain partitioning makes it possible to limit the total amount of processing power consumed when perform certain tasks. The most obvious use of partitioning is for frustum culling, which removes large chunks of geometry from the processing pipeline. The geometry can be removed because it is not within the view frustum and, therefore, does not contribute any viewable content to the rendered image. In a perspective projection, the view frustum is a volumetric space that is defined by the near and far

planes, as well as the viewing angles. Any mesh geometry contained within the bounds of the frustum will contribute content to the final rendered image. Initially, the terrain is a single, large mesh. In fact, the terrain mesh is often too large to fit within the view. By partitioning the terrain, it is possible to quickly eliminate large chunks of data from the processing pipeline. In our system, partitions are defined by a rectangular volume, which can be used to test whether the partition is entirely inside of, partially inside of, or totally outside of the view frustum. In the case where the partition is entirely outside of the view, the partition is neither processed nor rendered. Used in this manner, partitioning improves the execution speed without any negative impacts.

5.4.1.2 Level of detail

In order to support reasonably large terrains, the system employs a custom level of detail technique that integrates seamlessly with the deformation strategy. The technique is used to reduce the total number of polygons processed by the graphics processing pipeline, while preserving the visual fidelity of the scene. Given the current view, the technique will selectively render different resolution instances of each tile, thereby improving performance. It is a hybrid approach that combines a number of different practices derived from known solutions to define a custom method. In particular, the technique combines elements from [1, 4, 49, 53]. Elements from each are integrated into a single algorithm that is simple, yet powerful.

The technique uses the partitions defined by the spatial partitioning strategy as unique terrain tiles. Most of the computational burden for the technique is assumed during initialization, when static vertex and index buffers for data are constructed and

stored. These buffers are used throughout program execution, which greatly reduce the impact to runtime since minimal further computation is required.

During initialization, each tile of the partitioned terrain is prepared for runtime use. For each tile, a set of lower resolution instances are generated by sampling every other vertex from its immediate higher-resolution, parent mesh. The approach for sampling is the same as in Geomipmapping. However, unlike Geomipmapping, this approach uses skirts instead of stitching to prevent cracking at the edges where neighbors are currently being rendered at different resolutions. Also, a set of blending values are computed for Geomorphing. Geomorphing is used to eliminate popping artifacts that can be perceived as tiles change from one resolution to the next. At runtime, the blending values are used in conjunction with the currently active tile instance and the viewing location to compute a displacement for each vertex. The displacement replaces the instantaneous movement of vertices with a gradual one that is less noticeable. The goal in devising the solution as we have, was to place the majority of the workload in the preprocess step, which minimizes the overhead assumed at runtime.

During runtime, each partition in the terrain is evaluated and rendered independent of each other. In order to further reduce complexity, the solution borrows its approach for selecting the correct mesh instance from Geoclipmapping. The camera position is used to define the center of the region. A lookup table of radii is used to specify the correct resolution instance of each tile by evaluating the distance from the camera to the tile. Once the correct instance is decided on, the instance data for the tile is submitted for processing (Figure 26). The data submitted includes vertex positions, normals, texture coordinates, texture data, blending values, and skirts. It is even possible

to store all of the mesh data statically on the GPU, instead of physically transferring it from system memory to video memory each frame. As a result, the solution can produce high quality terrain imaging with a reduced total performance cost achieved through the hybridization of various methods to produce a fast and efficient multi-resolution strategy.

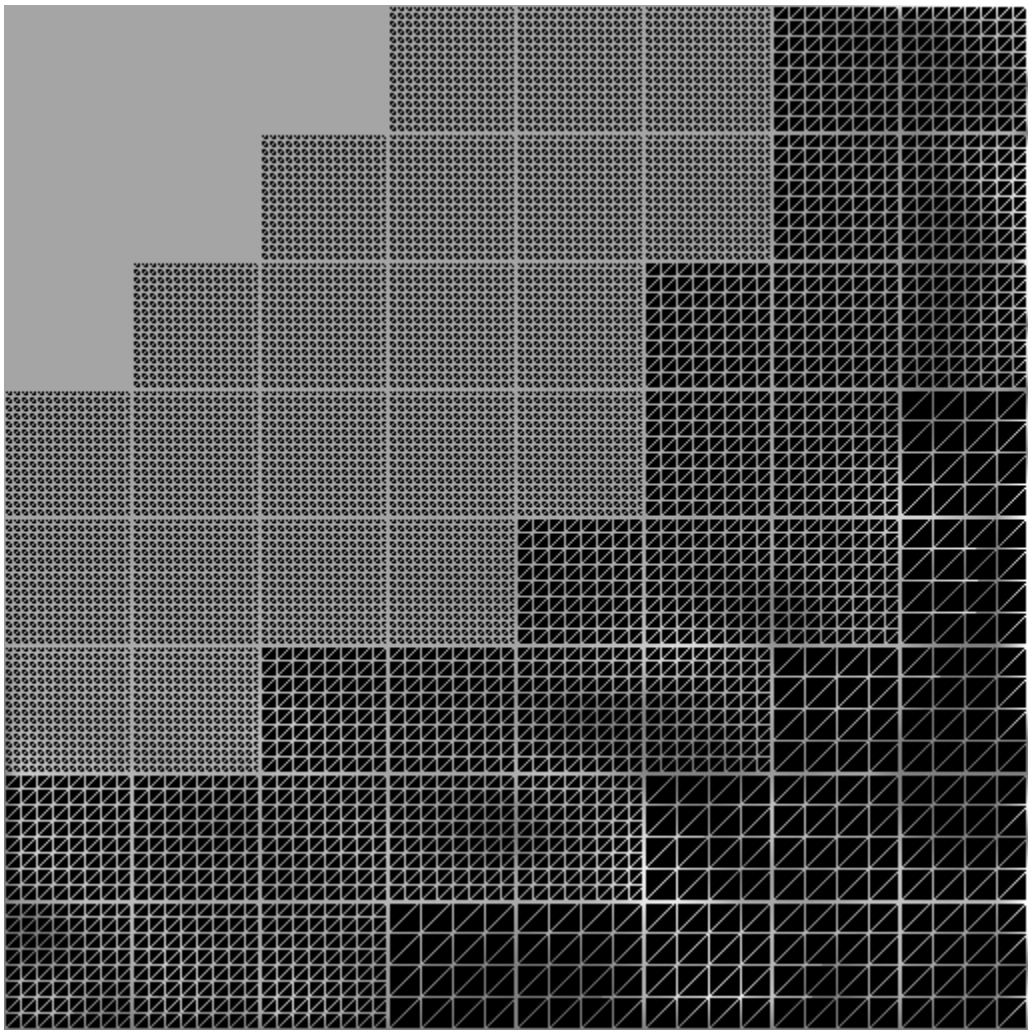


Figure 26 Orthographic view of the terrain with level of detail active.
The level of detail technique is used to identify and submit the correct resolution instance for processing using the location of the camera (the upper-left corner in this case).

5.4.2 Rendering Components

The focus of the system was to demonstrate the use of the component framework and to display the deformation strategy in action. Consequently, the components

incorporated for rendering were kept simple. Although elementary techniques were used, more elaborate approaches could be substituted to achieve more complex effects.

The Rendering component is comprised of solutions for texturing, shading, and atmospheric effects. For texturing, the system uses a tiled texture to represent a grassy ground-cover. Tiled texturing is simple, yet effective for our demonstrative purposes. For shading, the texture is blended according to the surface normals to create a simple surface-illumination effect. Since real-time shading of terrains is a complex field that can easily overtax the runtime, simple shading was used to limit its impact on the application. For atmospheric effects, the solution is only capable of rendering fog. Fog is commonly supported as a hardware feature, which makes it an obvious choice for inclusion because of our focus on exploiting GPU features. Once again, other types of atmospheric effects were excluded to limit total impact on runtime performance. The decision to use simple methods to suffice for rendering is not indicative of system limitations. Instead, these methods were selected because they compliment the system design by accomplishing their goals without infringing on the greater goals of the system.

5.4.3 Animation Components

Animation is concerned with the motion of objects and their subparts in the virtual world. In terrain visualization, the Animation component addresses needs of both the terrain and the virtual objects. The terrain system is an interactive entity in the virtual world; therefore, it must provide the facilities necessary for interfacing with external objects. The Animation tasks address the most fundamental activities and functional features required for dynamism and interaction. Collision detection ensures that external objects can query the terrain in order to determine proper location and to prevent

erroneous interpenetrations. Motion control is used to direct the dynamism of the terrain as its surface is deformed according to the rules of engagement for the solution.

Both collision detection and motion control features are included in the solution. Collision detection for the terrain is achieved through the provision of a query method. The function is used to compute the height y of the terrain at any given (x, z) location, which is useable by external objects. Objects may query the height to detect collisions and to determine where surface contact occurs between the object and the terrain. With respect to dynamic terrain, the computed height value is the starting point from which the object may sink itself into the ground for simulation and visualization purposes. The depth of penetration is recompensed by the deformation strategy which displaces vertices beneath the object to remove visible surface penetrations. The concept is that object-terrain penetrations (i.e. collisions) are intentionally created and then compensated in subsequent steps by deforming the terrain.

Vertex displacement for the terrain executes under the guidance of the motion control model of the solution. Out of necessity, the motion control in this solution enforces a constrained model where the terrain's positional vertices can only move perpendicularly to the ground plane. The constrained model is necessary to preserve the evenly-spaced, rectilinear layout of the elevation data. As objects impose forces on the terrain, the vertices compact and rebound to deform its surface. The constrained model is implicitly enforced by the deformation technique strategy previously presented. Specifically, the implementation of the technique in our system uses a multi-pass render-to-texture on the GPU to compute the displacement offsets. Using the depth buffer to compute the penetration depth of the object into the terrain provides the vertical

displacement that, in turn, preserves the regularity of the data according to the motion control model.

Together, the Animation tasks provide the functionality necessary to create an interactive, dynamic terrain solution. Collision detection provides the starting point for referentially placing objects on the terrain surface. After the simulation system adjusts the placement of the object, the motion control model that is inherent to the deformation technique guides the displacement of vertices to create the illusion of object-surface interactivity. Together, the Animation tasks provide the means for independent entities in the scene to detect and interact with the terrain.

5.4.4 Application Logic/Application-Specific Features

In general, terrain visualization systems are designed and developed for use as a subsystem within an application that has a greater purpose. In our solution, the terrain system was designed for use in off-road ground vehicle simulation and visualization. The terrain system makes uses application-specific features that are implemented for use by the ground-vehicle solution. In particular, the solution provides facilities for the physics-based simulation model of the vehicle to interact and affect the terrain directly. For instance, the option to associate additional per-vertex information is included so that elemental soil properties can be loaded, and subsequently read, from localized areas of the terrain. The feature could be used by the simulation system to compute simulation-relevant values such as tire-soil cohesiveness. Obviously, the cohesion of a tire on the terrain is not of any concern to the terrain solution; however, the system might not be considered complete without it presence. Considering the potential for future research, the terrain system could be extended in the future to make use of this capability. It is also

assumed that as the project evolves more application-specific features will be added to improve the terrain system's functional capabilities.

Development of a complete terrain visualization solution is not a trivial undertaking, but the component framework eases the design through its high-level abstraction of potential system features and functionality. Our research used the component framework in the specification of the terrain system for use in an off-road ground vehicle visualization and simulation. The framework offers direction and guidance in the creation of a terrain solution to establish parallels and differences between our solution and others. In addition, the framework eases the design by specifying those pieces necessary for accomplishing specific goals that would be unclear otherwise. As more researchers specify terrain visualization techniques and systems in the context of the component framework, comparative and collaborative work will conspire to improve the field.

5.5 Technical Details

The terrain system was built for inclusion as a subsystem in the simulation and visualization system for off-road ground vehicles. In particular, the solution was intended to compliment a physics-based vehicle simulation, by offering more visual cues regarding the activities and motion of the vehicle under non-ideal driving conditions. In order to cooperate with the rest of the system, the solution was built under certain technical constraints and specifications. The solution was developed in a Microsoft Windows environment using C++ and the OpenGL 2.0 API. In OpenGL 2.0, the GL Shading Language is available through extensions, which was used to access the programmable pipeline on the GPU. In addition, the render-to-texture features used for deformation

routine were achieved through the use of the framebuffer object extensions. Together, these features provided all of the facilities necessary for the terrain system.

A major differentiator from the solution we developed and other systems is the inclusion of the deformation technique using the programmable pipeline to exploit the graphics hardware. For clarity, the following documents some of the technical details of the deformation technique, to assist others in reproducing our work.

As previously noted, the deformation technique uses the OpenGL framebuffer object (FBO) extension for render-to-texture functionality. For our system, a single FBO is used to store the entire terrain in video memory because we imposed artificial limits on the total terrain size. However, one FBO per tile could be used by applications requiring larger terrains. Each frame, the initial offsets are computed in the areas where the vehicle(s) are located by configuring the FBO to use the depth buffer capability of the GPU. The algorithm for computing the initial offsets is as follows:

1. Query the height(s) of the contact points for the vehicle on the terrain and position the vehicle accordingly.

2. Execute the application-specific simulation model to sink the vehicle contact points into the terrain.

3. Configure the view properties to the view as explained in Section 4.2.2.

4. Compute the offsets by rendering to the FBO/render-target.

    a. Render the terrain tiles encapsulating the vehicle.
    b. Render the vehicle.

        i. In the fragment shader, compute the depth difference between the vehicle and the terrain.
        ii. Store values where the vehicle is penetrating the terrain in an offset map.

5. Restore the previous camera view.

Once the initial offsets are computed, we enact a simple and effective redistribution process to the offset map. Since the offset map is stored as a texture, our solution applies a box-filter derivative to smooth away jagged edges. The filter is encoded in the fragment shader used to generate the offset map, which allows full control of pixel processing. Although the process we use for redistribution here is not physics-based, it is conceivable to use a more elaborate fragment shader to simulate soil mechanics.

After all of the offsets are computed, the terrain is ready to be processed and rendered. Once again, the programmable pipeline is exploited to draw the terrain. In the vertex shader, the terrain elevation data is translated, rotated and scaled appropriately. Using the (x, z) coordinates of the vertex, texture coordinates are derived and used to lookup the redistributed offset value in the offset map. The vertex texture lookup is a feature of Shader Model 3.0 that makes displacement mapping on the GPU possible. The original height, the redistributed offset, and the relevant Geomorphing values are used to compute the position of the vertex for the current frame. Next, vertex colors and texture coordinates are adjusted based on the associated offset of the vertex, to create an illusion of shading where tire tracks have carved into the surface. The results are passed onto the fragment shader, which renders the geometry in a straightforward manner by applying textures, fog, and colors on a per-pixel basis.

The final rendered image displays a tire track carved into the terrain surface where the vehicle has traversed (Figures 27 and 28). The remnants of the vehicle's driving path enrich the scene with visual cues and detail that improve the visual quality of

the image. Without the tire markings, the terrain would appear unrealistically rigid. Furthermore, visually recounting the movement of the vehicle is impossible.

(a)

(b)

Figure 27 The vehicle deforms the terrain.
The vehicle deforms the terrain dynamically as it moves (a) and carves tire tracks (b).

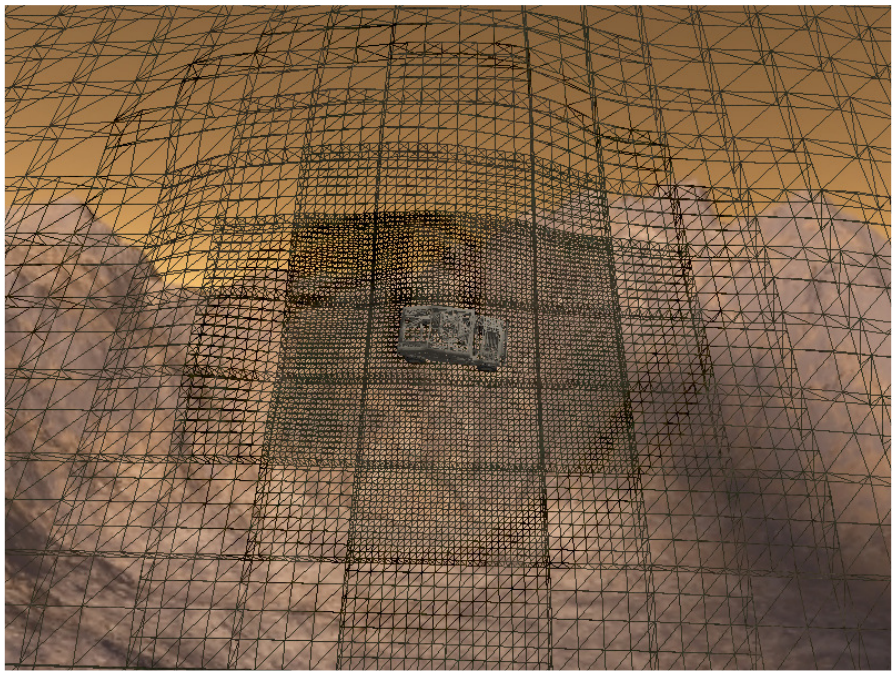


Figure 28 The integrated level of detail solution.
The level of detail system is coordinated to work with the deformation technique.

5.6 Test Results

The off-road ground-vehicle system was tested to assess the performance impact of the deformation technique with respect to the rest of the system. The specifications of the computer used in the test can be found in Table 1. Two different terrain sizes were tested and the results are documented in Table 2.

Processor	Intel Xeon 3.00GHz (Dual Processor)
RAM	2 GB
Video Card	NVIDIA GeForce 7800 GT, 256MB GDDR3 RAM, PCI-Express
Operating System	Windows XP SP2

Table 1. Test Machine Specifications

65m x 65m			129m x 129m		
1m posts	.25m posts	.125m posts	1m posts	.25m posts	.125m posts
480 fps	230 fps	75 fps	470 fps	220 fps	68 fps

Table 2. Experimental Test Results.

The results are very promising as frame rates for the solution were maintained well above the target 60 frames per second required of real-time graphics applications. As expected, performance drops as the resolution of the mesh is increased by the DDR subdivision of the terrain. However, an excess amount of terrain data would suggest the need for a more elaborate level of detail technique, the inclusion of a paging strategy, or in-memory data compression. As previously noted, the integration of dynamism with such techniques is beyond the scope of this initiative, and left as an open question for future research. Regardless, the data suggests that the deformation technique itself has a limited impact on the overall solution, which supports our assertion that current hardware is well-suited for and capable of supporting dynamic terrain.

6. Conclusion and Future Work

6.1 Conclusion

Currently, most research in the field of Computer Graphics is heavily geared towards producing photorealistic imagery, which can be computationally expensive. At the same time real-time computer graphics systems must maintain a minimal framerate, while achieving high-quality imagery. The conflict of interest between realism and performance is the primary point of contention in terrain visualization, which strives to visualize high-fidelity terrains with a nominal impact to performance. Recent advances in hardware afford the option to improve terrain visualization systems by improving realism without hindering performance.

Static and dynamic terrains are the two approaches in terrain visualization and there are benefits and drawbacks with each. Static terrain uses a fixed, rigid terrain mesh to model the landscape, whereas dynamic terrain supports runtime surface deformation of the terrain model. Consequently, static terrains may only simulate hard surfaces, while dynamic terrains can represent any surface type.

Currently, static terrain solutions dominate the field. The dominance arose from the development and widespread acceptance of techniques for static terrain that were developed based on concerns regarding hardware in the past. The rigid nature of a static terrain allows for many algorithmic optimizations that make it possible to render large terrain meshes in real-time. The concerns regarding performance that drove these earlier techniques indirectly served to entrench rigid terrain as the foremost solution. While

static terrain is often an acceptable solution, it is far from a complete solution, due to its inability to represent non-rigid terrain.

Dynamic terrain systems can achieve a truly realistic terrain simulation and visualization. Unlike static terrain, dynamic terrain has the potential to support the full range of surface hardness, allowing external forces to mold its shape. The drawback of dynamic terrain is the added workload associated with extra processing and rendering tasks necessary for terrain dynamism. While a static terrain approach was the best option in previous years, computer hardware has improved greatly. These improvements afford researchers the opportunity to pursue new and innovative approaches in terrain visualization, including techniques for real-time dynamism.

6.2 Contributions

With regard to terrain visualization, a number of innovations are presented throughout the thesis that serves to improve the field. These innovations bolster the prospect of improving terrain visualization through guiding research focus and promoting the use of dynamism in modern terrain visualization systems.

6.2.1 Component Framework for Terrain Visualization

The first major innovation of this research is the Component Framework for Terrain Visualization, which defines a unifying architecture for terrain solutions. In addition, systems using the common architecture and componentized structure share a singular baseline that makes it possible to perform comparative analysis on otherwise disparate works. The framework is structured according to common tasks in Computer Graphics applications to provide a sensible construct that is both flexible and complete for designing a terrain system. The high-level tasks are further decomposed into more

focused components that address the various challenges faced in terrain visualization. The distinction between components guides research directions, keeping focus on those topics within the problem domain of a specific component. By breaking down the system into its set of components, the development of these highly complex systems is eased. At the same time, the framework identifies relationships between the unique components, binding them into a collective whole. The relationships aid in diagnosing compatibility issues between algorithms employed for unique, yet related, tasks. Lastly, the framework supports adaptation and extensibility for the future, which is necessary as the field evolves. The framework is designed to evolve with the field, allowing new components to be introduced and old ones exorcised as requirements change. To demonstrate the practical nature of the framework, it was used in the design and development of the terrain system presented in Chapter 5. The component framework for terrain visualization improves the field through its structured, consistent approach for describing terrain systems, making it suitable for practical use and research purposes.

6.2.2 Dynamic Terrain

The second major focus of the thesis is in championing the use of dynamism in terrain visualization. Specific innovation in dynamic terrain include: developing the Dynamically Divisible Regions, specifying the tasks necessary for processing dynamic terrain, and originating a new technique for terrain deformation using the GPU.

6.2.2.1 Dynamically Divisible Regions

The first issue in dynamic terrain discussed is the potential for a terrain mesh to offer an inadequate mesh resolution with respect to the object(s) in the scene that impose the deformation. The concern is addressed through the presentation of the Dynamically

Divisible Region (DDR), which is a subdivision surface specification useful in the visualization of dynamic terrain surfaces. The DDR is derivative of Dynamic Extension to Resolution (DEXTER) [30]; however, a DDR defers implementation and offers adaptability that makes it suitable for use with multi-resolution techniques other than Hierarchical Bin-Tree methods. As a demonstrative means, Section 4.1 presents a specialized DDR technique that makes explicit use of the GPU to increase the mesh resolution as needed by the deformation solution. This solution was also implemented in by the system presented in Chapter 5 to generate a higher resolution mesh.

6.2.2.2 Terrain Deformation Technique

The second topic in dynamic terrain addressed herein was the identification and specification of four high-level tasks required to deform, process, and render a dynamic terrain. Previous works in the area focus on developing a specific algorithm that displaces local vertices according to a physics-based or appearance-based control scheme. Although it is important, the issue of vertex displacement is not sufficient as a complete dynamic terrain solution. In contrast, the thesis covers the entire process through a four step method that constitutes the definition of a complete solution. The process description for terrain dynamism identifies distinct tasks that contribute to a successful solution without introducing limiting compulsory requirements. As such, the procedure is a generic approach for dynamic terrains that can be specialized to support a multitude of simulation types, which translates to just as many possibilities for visualization purposes.

Extrapolated from the development of our solution for dynamic terrain, we present a novel approach to terrain deformation using the GPU. Notably, the algorithmic steps to deform the terrain and display the results span across four high-level tasks. The

technique relies on the ability to transform the Regular Grid Network of elevation data from system memory to texture memory and, subsequently, processing that data within a programmable pipeline that supports Shader Model 3.0. The first step, displacement, is achieved by using the depth buffer functionality of the GPU to record interpenetrations of the surface by non-terrestrial objects, which begets the initial Offset Map. The Offset Map represents the displacement of vertices over time as imposed by the object. The second step, redistribution, is executed on offsets values through the application of a custom filter that achieves a visual imitation of granular soil displacement. Redistribution adjusts a vertex' displacement value according to the soil simulation model, as it is implemented in the filter. Optionally, periphery data like normals, textures, and vertex-colors are updated to coincide with the newly deformed terrain. Lastly, the data is submitted for processing and rendering, using the programmable pipeline to displace, deform, color, shade, and texture the terrain geometry, which outputs the final image to the screen. By incorporating the technique in a terrain system, it is possible to visualize interactive terrain dynamics while minimizing the impact on the runtime by offloading the majority of the work to the GPU.

Dynamic terrain visualization is a better solution than static terrain visualization, and we present a foundation for developing terrain systems that support dynamism. The component framework is a universal tool for terrain visualization researchers and developers that defines and unifies various concerns and focuses within the domain of terrain visualization. In an effort to promote the use of dynamic terrain, the research presents a formalization of terrain dynamism through the specification and description of four high level tasks. To further encourage the use of terrain dynamics in practice, a

novel technique for GPU-based deformation and soil simulation is proposed. Together, these innovations serve to advance terrain visualization into a new era of visual realism that enhances the system and improves the user experience.

A variety of people, corporations, and industries stands to benefit from the advancement in functionality and realism of systems for terrain visualization. Entertainment media, including video games and movies, generate billions of dollars in revenue every year. Savvy consumers have increasingly higher expectations of the media which they will pay for. Improving the environmental realism within the virtual world through interactive, reactive terrain is a monumental step of progression that can impress consumers and keep them coming back for more. Training and Simulation (T&S) is another lucrative application domain that makes extensive use of terrain visualization, but it is also one that saves lives. Military forces use Training and Simulation systems to train personnel in vehicular operations, tactical deployment, and reconnaissance missions that translate directly into real world activities. The goal is to provide a realistic, immersive experience that prepares trainees for intense situations without risking damage to expensive equipment or endangering human life. In fact, the desire for realistic flight simulators to train pilots is a major contributor to the popularity in research for terrain visualization. As third example, Geographic Information Systems (GIS) are focused on the sampling, analysis, and display of terrain data for the purpose of environmental research and commercial ventures, amongst others. The field is gaining notoriety as Global Positioning Systems (GPS) become more and more popular in various communities for tracking mobile entities like people and vehicles. Obviously, there is a

vested interested in improving the visualization of the terrain for all of these systems, because it directly impacts the effectiveness of the system.

6.3 Future Research Directions

The work completed within the thesis makes great strides in advancing terrain visualization; however, the solutions presented can be extended and improved upon with further research and development.

6.3.1 Paging Dynamic Terrain

In our system, we were not concerned with the issue of reading terrain data from an external storage device at runtime because we impose a superficial maximum terrain size. Paging solutions serve as the primary approach for handling the terrain data in systems requiring the display of very-large terrains. The primary challenge of terrain paging is overcoming the slow transfer times of data from external storage to faster main memory. In static terrain, a page can be assumed to work under a read-only, unidirectional flow because the data remains unchanged once it is in memory. In contrast, dynamic terrain is modifiable at runtime, which makes it a read/write model and, thereby, requires bidirectional, load-and-save data management. Since deforming the terrain modifies the elevation dataset, data modifications must be preserved as pages are released from main memory to ensure that reloading the data at a later time restores the altered surface. Preserving surface deformation requires the storage of the altered dataset back to the storage medium when the page of terrain is to be replaced. The need to perform write-backs complicates paging; however, preserving surface modifications to paged data is necessary for dynamism in very-large terrain visualization.

6.3.2 Methods for Vertex Attribute Maintenance

In the discussion of the tasks required for processing and rendering dynamic terrain, we point out the potential need to maintain vertex attributes. For the terrain system used in the off-road ground vehicle simulation, only vertex colors needed to be proactively updated. It is assumed as dynamic solutions become more common, more attributes will be identified that require maintenance. As these values are discovered, there will be a need to develop advanced methods for maintenance that make optimal use of available hardware. Pursuance of these methods will improve both runtime performance and visual fidelity. Some ideas we have considered include:

- The generation and maintenance of normal maps for regions using the GPU, especially as the region undergoes runtime deformation.

- The creation of dynamic procedural texture maps that can be reconfigured as deformation occurs to visually support surface changes, such as 'dirtying' the impact area of a soft terrain.

- The preservation of per-vertex soil properties that change as the vertex is displaced by external forces. Potential properties include cohesion, weight, and compressibility that could be used by the soil simulation model when displacing vertices.

6.3.3 Physics-based Simulation

Lastly, there are a many possibilities for improving the dynamism element in terrain visualization with the advent of more elaborate and meaningful simulation models. Primarily, we are concerned with evolving physics-based simulations for the soil and objects interacting with the soil. Physics-based simulations make use of the physical laws to guide the motion of objects and surfaces in the scene, resulting in a more realistic and accurate representation.

6.3.3.1 Improved Granular Soil Models

Dynamic terrains that use an underlying physics-based simulation offer a more realistic depiction of surface interaction. In the future, we intend to develop a simulation model derived from Soil Mechanics that executes as part of the deformation technique using the GPU. Specifically, the redistribution process will be specialized to make use of a physics-based model akin to [54] and [55]. For systems that are more concerned with performance than simulation accuracy, it is possible to develop more elaborate and meaningful appearance-based methods, like [56] and [58], for redistribution than the box-filter smoothing we used here.

6.3.3.2 Extend for use with Terramechanics

As a practical application using dynamic terrain system, future work will incorporate terrain dynamism with physics-based vehicular modeling that runs in real-time. The innovation of this work is the inclusion of a Terramechanical model to produce a very realistic and accurate ground-vehicle simulation. Terramechanics is the study of tire-soil interaction, focusing on the physical laws and principles that govern land locomotion and mobility. Terramechanical simulation requires a physics-based simulation of the interactive terrain surface model and the vehicular modeling. Thus, the work in dynamic terrain is a necessary component to develop a complete visual system that includes Terramechanics.

6.4 Final Words

As time progresses, it is expected that hardware advancements will provide more processing power to be used in the software domain. It is wise to take advantage of available processing power by incorporating new features that serve to improve the

system. Terrain visualization systems are an important subsystem in a many software applications across a variety of problems domains. Those systems that employ a terrain solution are improved through the advancement of simulated and visual realism afforded by the terrain visualization solution. The underlying theme behind our research is to improve terrain visualization by encouraging a more robust and accurate representation of terrain in the visual system through the use of dynamic terrain. The current initiative has promoted dynamism through its proposals of the component framework, the deformation specification, and the GPU-based deformation technique. Combined with improvements to hardware, these contributions improve terrain visualization realism and enhance the user-experience.

REFERENCES

[1] T. Ulrich, "Rendering Massive Terrains using Chunked Level of Detail Control," in *Course presented at the 29th annual conference on Computer graphics and interactive techniques*. San Antonio, Texas, 2002.

[2] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, S. Worley, W. Mark, and J. Hart, *Texturing & Modeling: A Procedural Approach*, Third Edition ed: Morgan Kaufmann Publishers, An Imprint of Elsevier Science, 2002.

[3] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communications of the ACM*, vol. 19, pp. 547-554, 1976.

[4] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids," *ACM Transactions on Graphics*, vol. 23, pp. 769-776, 2004.

[5] P. Lindstrom and V. Pascucci, "Visualization of large terrains made easy," in *Proceedings of the 12th conference on Visualization*. San Diego, California: IEEE Computer Society, 2001.

[6] J. Blow, "Implementing a Texture Caching System," in *Game Developer Magazine*, 1998, pp. 46-56.

[7] R. L. Ferguson, R. Economy, W. A. Kelly, and P. P. Ramos, "Continuous level of detail for visual simulation," in *Proceedings of the 1990 IMAGE V Conference*. Tempe, Arizona, 1990, pp. 144-151.

[8] H. Hoppe, "Progressive meshes," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*: ACM Press, 1996.

[9] D. Wagner, "Terrain Geomorphing in the Vertex Shader," in *ShaderX2 - Shader Tips & Tricks*, W. F. Engel, Ed.: Wordware Publishing, Inc., 2003.

[10] E. E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces.," in *Ph.D. Thesis, Dept. Computer Science*: University of Utah, 1974.

[11] C. M. Goral, K. E. Torrance, G. D. P., and B. Battaile, "Modeling the interaction of light between diffuse surfaces," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*: ACM Press, 1984.

[12] T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, vol. 23, pp. 343-349, 1980.

[13] N. Hoffman and K. Mitchell, "Real-Time Photorealistic Terrain Lighting," in *Proceedings of the 2001 Game Developers Conference*, 2001.

[14] A. J. Stewart, "Fast Horizon Computation at All Points of a Terrain With Visibility and Shading Applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, pp. 82-93, 1998.

[15] L. Williams, "Casting curved shadows on curved surfaces," in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*: ACM Press, 1978.

[16] F. C. Crow, "Shadow algorithms for computer graphics," in *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. San Jose, California: ACM Press, 1977.

[17] M. McGuire and P. Sibley, "A Heightfield on an Isometric Grid," in *Sketch presented at the 31st annual conference on Computer graphics and interactive techniques*, 2004.

[18] R. S. Nielsen, "Real Time Rendering of Atmospheric Scattering Effects for Flight Simulators," in *Master's Thesis, Informatics and Mathematical Modelling*: Technical University of Denmark, 2003.

[19] A. J. Preetham, P. Shirley, and B. Smits, "A Practical Analytic Model for Daylight," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*: ACM Press, 1999.

[20] R. A. Finkel and J. L. Bentley, "Quad Trees, a Data Structure for Retrieval on Composite Keys.," *Acta Informatica*, vol. 4, pp. 1-9, 1974.

[21] M. Pritchard, "Direct Access Quadtree Lookup," in *Game Programming Gems 2*, M. Deloura, Ed. Hingham, Massachusetts: Charles River Media, 2001, pp. 394-401.

[22] T. Nuydens, "Terrain texturing," http://www.delphi3d.net/articles/viewarticle.php?article=terraintex.htm, 2002.

[23] T. Polack, *Focus On 3D Terrain Programming*, 1st Ed. ed: Course Technology PTR, 2002.

[24] G. Snook, *Real-Time 3D Terrain Engines Using C++ and DirectX 9*, 1st Ed. ed: Charles River Media, 2003.

[25] C. Bloom, "Terrain Texture Compositing by Blending in the Frame-Buffer." http://www.cbloom.com/3d/techdocs/splatting.txt, 2000.

[26] H. Hoppe, "View-dependent refinement of progressive meshes," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*: ACM Press/Addison-Wesley Publishing Co., 1997.

[27] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. urner, "Real-time, continuous level of detail rendering of height fields," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*: ACM Press, 1996.

[28] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes," in *Proceedings of the 8th conference on Visualization*. Phoenix, Arizona, United States: IEEE Computer Society Press, 1997.

[29] A. A. Pomeranz, "ROAM using Surface Traingle Clusters (RUSTiC)," in *M.S. Thesis, Department of Computer Science*: University of California (Davis), 1998, pp. 42.

[30] Y. He, "Real-time visualization of dynamic terrain for ground vehicle simulation," in *Ph.D. Thesis, Department of Computer Science*. Iowa City: University of Iowa, 2000, pp. 165.

[31] M. Garland and P. S. Heckbert, "Fast Polygonal Approximation of Terrains and Height Fields," Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 CMU-CS-95-181, September 19, 1995 1995.

[32] L. D. Floriani, P. Magillo, and E. Puppo, "Multiresolution Models for Topographic Surface Description," *The Visual Computer*, vol. 12, pp. 317-345, 1996.

[33] D. Luebke and C. Erikson, "View-dependent simplification of arbitrary polygonal environments," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*: ACM Press/Addison-Wesley Publishing Co., 1997.

[34] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level of Detail for 3D Computer Graphics*: Morgan Kaufmann, 2003.

[35] Y. He, "Multiple Resolution Surface Representations and Their Extension for Dynamic Terrain: A Survey," Survey Paper, Department of Computer Science, The University of Iowa, 1999.

[36] J. El-Sana and A. Varshney, "Generalized View-Dependent Simplification," presented at Proceedings of Eurographics, 1999.

[37] D. Cohen-Or and Y. Levanoni, "Temporal Continuity of Levels of Detail in Delaunay Triangulated Terrain," *Proceedings of the 7th conference on Visualization*, pp. 37-42, 1996.

[38] P. Cignoni, E. Puppo, and R. Scopigno, "Representation and Visualization of Terrain Surfaces at Variable Resolution," *The Visual Computer*, vol. 13, pp. 199-217, 1997.

[39] H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *Proceedings of the 9th conference on Visualization*. Research Triangle Park, North Carolina, United States: IEEE Computer Society Press, 1998.

[40] M. P. Kumler, "An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs)," *Cartographica*, vol. 31, pp. 1-48, 1994.

[41] A. Ogren, "Continuous Level of Detail in Real-time Terrain Rendering," in *M.S. Thesis, Computing Science*. Umea, Sweden: University of Umea, 2000.

[42] B. Turner, "Real-Time Dynamic Level of Detail Terrain Rendering with ROAM." Gamasutra: http://www.gamasutra.com/features/20000403/turner_01.htm, 2000.

[43] M. Duchaineau, "ROAM Algorithm Version 2.0," http://www.cognigraph.com/ROAM_homepage/ROAM2/, 2003.

[44] S. Röttger, W. Heidrich, P. Slussallek, and H. P. Seidel, "Real-Time Generation of Continuous Levels of Detail for Height Fields," in *Proceedings of 1998 International Conference in Central Europe on Computer Graphics and Visualization*, 1998.

[45] P. Lindstrom and V. Pascucci, "Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, pp. 239-254, 2002.

[46] J. Levenberg, "Fast View-Dependent Level-of-Detail Rendering Using Cached Geometry," in *Proceedings of the 13th conference on Visualization*: IEEE Computer Society Press, 2002.

[47] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno, "BDAM - Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization," *Computer Graphics Forum*, vol. 22, pp. 505-514, 2003.

[48] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno, "Planet-sized Batched Dynamic Adaptive Meshes (P-BDAM)," *Proceedings of the 14th conference on Visualization*, pp. 147-155, 2003.

[49] W. H. de Boer, "Fast Terrain Rendering Using Geometrical MipMapping," http://www.flipcode.com/articles/article_geomipmaps.shtml, 2000.

[50] L. Williams, "Pyramidal Parametrics," in *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*. Detroit, Michigan, United States: ACM Press, 1983.

[51] L. E. Hitchner and M. W. McGreevy, "Methods for user-based reduction of model complexity for Virtual Planetary Exploration," presented at IS&T/SPIE's Symposium on Electronic Imaging, Proceedings of SPIE, San Jose, 1993.

[52] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shanz, "Designing a PC Game Engine," *IEEE Computer Graphics and Applications*, vol. 18, pp. 46-53, 1998.

[53] A. Asirvatham and H. Hoppe, "Terrain Rendering using GPU-based Geometry Clipmaps," in *GPU Gems 2*, M. Pharr and R. Fernando, Eds.: Addision-Wesley, 2005.

[54] X. Li and J. M. Moshell, "Modeling soil: realtime dynamic models for soil slippage and manipulation," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*: ACM Press, 1993.

[55] B. Chanclou, A. Luciani, and A. Habibi, "Physical Models of Loose Soils Dynamically Marked by a Moving Object.," *Computer Animation*, pp. 27-35, 1996.

[56] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins, "Animating Sand, Mud, and Snow," *Computer Graphics Forum*, vol. 18, 1999.

[57] K. Onoue and T. Nishita, "Virtual Sandbox," in *11th Pacific Graphics Conference on Computer Graphics*, 2003.

[58] K. Onoue and T. Nishita, "An Interactive Deformation System for Granular Material," *Computer Graphics Forum*, vol. 24, pp. 51-60, 2005.

[59] W. Pan, Y. E. Papelis, and Y. He, "A Vehicle-Terrain System Modeling and Simulation Approach to Mobility Analysis of Vehicles on Soft Terrain," in *Proceedings of the International Society for Optical Engineering*, vol. 5422, G. R. Gerhart, C. M. Shoemaker, and D. W. Gage, Eds. Bellingham, Washington USA, 2004.

[60] X. Cai, F. Li, and S. Zhan, "Research of Dynamic Terrain Based on Regular Triangles in Complex Battlefield Environments," *Journal of System Simulation*, vol. 17, 2005.

[61] X. Cai, F. Li, H. Sun, and S. Zhan, "Research of Dynamic Terrain in Complex Battlefield Environments," in *Edutainment 2006, Lecture Notes in Computer Science 3942*: Springer-Verlag, 2006.

[62] G. Chen and J. Zhang, "Dynamic Terrain LOD with Region Preservation in 3D Game Engine," in *Edutainment 2006, Lecture Notes in Computer Science 3942*: Springer-Verlag, 2006.

[63] A. Shamir, V. Pascucci, and C. L. Bajaj, "Multi-Resolution Dynamic Meshes with Arbitrary Deformations," in *Proceedings of the 11th conference on Visualization*: IEEE Computer Science Press, 2000.

[64] S. Kircher and M. Garland, "Progressive multiresolution meshes for deforming surfaces," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Los Angeles, California: ACM Press, 2005.

[65] Y. He, J. Cremer, and Y. Papelis, "Real-time Extendible-resolution Display of On-line Dynamic Terrain," in *Proceedings of the 2002 Conference on Graphics Interface*. Calgary, Alberta, Canada, 2002.