An Integrated Global GIS and Visual Simulation System

Peter Lindstrom David Koller William Ribarsky
Larry F. Hodges Augusto Op den Bosch Nick Faust

Graphics, Visualization, & Usability Center
Georgia Institute of Technology

Abstract

This paper reports on an integrated visual simulation system supporting visualization of global multiresolution terrain elevation and imagery data, static and dynamic 3D objects with multiple levels of detail, non-protrusive features such as roads and rivers, distributed simulation and real-time sensor input, and an embedded geographic information system. The requirements of real-time rendering, very large datasets, and heterogeneous detail management strongly affect the structure of this system. Use of hierarchical spatial data structures and multiple coordinate systems allow for visualization and manipulation of huge terrain datasets spanning the entire surface of the Earth at resolutions well below one meter. The multithreaded nature of the system supports multiple windows with independent, stereoscopic views. The system is portable, built on OpenGL, POSIX threads, and X11/Motif windowed interface. It has been tested and evaluated in the field with a variety of terrain data, updates due to real-time sensor input, and display of networked DIS simulations.

1 INTRODUCTION

This paper reports on significant progress in our efforts to design and construct a real-time visual simulation and geographic information visualization system, named VGIS (Virtual Geographic Information System) [17, 30]. VGIS supports the accurate depiction of terrain elevation and imagery, in addition to features such as ground cover and trees, moving vehicles, buildings and other static objects, roads, and atmospheric effects. Thus an entire environment composed of heterogeneous parts must be simulated and integrated at rendering time. The system must be set up to efficiently manage this integration and, ultimately, to manage dynamically the complexity of each part with respect to the others in order to both conform to a strict time budget and to present the most telling details. Integration of all this with a GIS database is important because many applications require access to geographically located information (e.g. building names, contents, and even floor plans). The GIS data can also be handled by the VGIS data managers and threaded through the real-time renderer for visualization in the 3D display environment.

The visual simulation system described above implies very large, even huge amounts of data. Automatic paging and caching techniques handling heterogeneous data from the different parts of the system must be in place. If, for example, the system is to visualize urban scenes, it must manage hundreds to thousands of buildings, plus their textures, and also street layouts. For flexibility the terrain visualization sub-system should handle terrain from any part of the world and integrate these terrains into a common coordinate system without seams or gaps (e.g. between levels of detail or due to multiple coordinate systems). All this should be in a hierarchical organization structure so that the terrain detail can be continuously adapted based on user viewpoint and scene content. Yet the hierarchy must be flexible so that detail can be added or deleted as needed. Such flexibility is quite important due to database size as the global datasets used with VGIS often require ten or more gigabytes.

In this paper we describe a visual simulation system that provides a structure supporting all the parts described above. We also discuss in detail our implementations for some of these parts, concentrating especially on global terrain visualization. The structure is in a multithreaded form to facilitate balanced and separable management of the system parts. It is also quite portable, due to standard libraries such as *Pthreads* and *OpenGL*, and has been ported to multiple workstation environments, including SGI and Sun platforms. We are now working on a PC version using Windows NT. The system has wide applicability, having been used for battlefield visualizations, tactical planning, and complex urban visualizations.

2 RELATED WORK

VGIS takes advantage of advances in a number of areas to create an integrated global geographic information visualization system. A large body of previous work has addressed issues in modeling, representing, and manipulating spatial data for geographic information systems. Applying 3D visualization techniques to global and spatial data has recently enjoyed increasing attention [12, 22].

VGIS manages its huge, complex terrain and GIS datasets in an efficient manner by using hierarchical spatial data structures. A number of such data structures have been adopted in GIS systems and other spatial databases. Samet [25] describes the quadtree, a fundamental data structure in the VGIS system. Representing spatial datasets that span the entire globe requires special data structures which take into account the curvature of the Earth and allow for efficient searching and rendering operations on the large amounts of data. Fekete [8] describes sphere quadtrees, a spatial data structure applicable to global representations of the Earth. Other researchers have proposed similar hierarchical spatial data structures which have been demonstrated to be useful for global geographic information systems [13].

VGIS also relies on multiresolution techniques to allow truly interactive visualization with its geometrically complex terrains. A large number of researchers have developed multiresolution representations and rendering techniques for large, complex terrains and height fields using polygonal meshes, as VGIS does. These algorithms attempt to represent surfaces within a given number of vertices, or within a given geometric error metric, or in a manner that preserves application specific critical features of the surface. Uniform grid methods or irregular triangulations are employed to represent the surfaces, and techniques including hierarchical subdivisions and decimations of the mesh are used for simplification and creation of multiresolution representations.

Much of the previous work on polygonalization of terrain-like surfaces has concentrated on triangulated irregular networks

(TINs). A number of different approaches have been developed to create TINs from height fields using Delaunay and other triangulations [10, 11, 27], and hierarchical triangulation representations have been proposed that lend themselves to usage in multiresolution level of detail algorithms [4, 5, 26]. Regular grid surface polygonalizations have also been implemented as terrain and general surface approximations [3]. Such a gridded terrain surface representation is used in VGIS and is described in [20]. Other surface approximation representations include techniques such as wavelet transforms [14] and methods that meet application specific criteria, such as preserving important terrain features [6, 10, 28].

VGIS uses an approach which treats the terrain as a single connected surface for rendering, using "continuous" level of detail representations for the terrain geometry. Similar methods for such "continuous" or "view-dependent" level of detail rendering for terrains and other surfaces are described in [3, 9, 20, 31, 32].

A number of visualization systems have been implemented which integrate 3D visualization techniques with large spatial geographic information and terrain data. Some systems stress accurate rendering of global images, or accurate modeling of environmental processes, often sacrificing interactivity of the system [21, 24]. Other systems emphasize tight integration of the 3D visualization with the powerful spatial analysis capabilities of GIS [7].

Systems such as VGIS place a high priority on real-time, highly interactive 3D visualizations of the spatial data. Maintaining truly real-time update rates in the face of large, complex datasets requires special techniques and time-critical visualization system designs. Bryson and Johan [1] discuss some issues particular to such time-critical computations in visualization environments. Software toolkits such as IRIS Performer [23] provide an architectural framework similar to that of VGIS which provides support for efficient rendering and simulation operations on high-end computer graphics workstations.

Interactive 3D visualization systems for visual simulation and training address many of the same technical problems as VGIS, including real-time rendering of large terrain databases. Flight simulator systems first implemented sophisticated image generation techniques to allow efficient rendering and visualization of complex spatial databases [33]. Recent developments have made interactive 3D graphics rendering of such databases possible using off-the-shelf graphics workstations. NASA's Virtual Planetary Exploration project [16] supported virtual exploration of planetary terrains such as Mars at interactive frame rates. SRI's TerraVision system also allows interactive viewing of 3D landscapes, using similar terrain level of detail and paging techniques as VGIS to allow very large, high-resolution geo-specific terrain datasets to be visualized. The T_Vision research project [15] provides a distributed virtual globe as a multimedia interface for visualizing geographic data. Other 3D terrain visualization systems use parallel architectures to render complex datasets while maintaining interactive performance for visual simulation and planetary visualization applications [2, 19].

3 SYSTEM DESIGN AND REQUIREMENTS

The rationale behind many of the design decisions made in the development of VGIS was driven by both hardware constraints and user requirements. Typical hardware constraints include available main memory, available texture memory, available precision in geometric calculations, rendering speed, disk transfer speed, etc., while the user requirements include interactivity, off-line processing time, display accuracy, data registration, as well as flexibility, extensibility, and portability from both the end user's and developer's perspectives.

Due to the large datasets that VGIS typically works with, most of the data must be put in secondary storage, and a paging scheme is used to bring in and cache the appropriate data for display. Level of detail techniques are applied to both geometry and imagery to further limit the amount of data stored in main memory and texture memory. These techniques are based on image quality metrics whose parameters can be manipulated interactively by the user to obtain a desirable balance between rendering speed and scene fidelity.

The user is given the ability to visualize the scene through multiple *views* which map onto separate windows. Each such view may display the scene from a different viewpoint, e.g. as a 3D immersive view or a 2D overview map, or may display different aspects of the same scene, e.g. as phototextured terrain or as a contour map with surface features such as roads and rivers turned on. In order to conserve memory, view-independent data is shared among the views and is accessed from a single primary cache.

To further obtain a high degree of interactivity, the system is broken down into a number of asynchronous threads that are prioritized according to their relevance to the final display update rate, which is one of the most important constraints in the system. For example, a dedicated render thread is used whose single task it is to update one or more views at the highest possible rate; level of detail (LOD) management is distributed over several threads according to the data they operate on, which generally update the scenes at a rate lower than the rendering rate; while a number of server threads execute only when data requests are made. This fine-grained subdivision of tasks ensures a high degree of CPU utilization and eliminates the bottlenecks often associated with blocking system calls (e.g. disk I/O, input device polling) in the real-time components of the system.

To accommodate data paging, level of detail management, and view culling, a quadtree data structure [25] is used to spatially subdivide and organize the terrain raster data. The globe is subdivided into a small number of pre-determined areas, each corresponding to a separate quadtree. The tiles associated with the quadtree nodes, or *quadnodes*, are stored in a file format that closely matches the internal representation, which allows for good paging performance. Rather than using a single global coordinate system, a large number of local coordinate systems is used. This is necessary as the precision afforded by current graphics hardware is insufficient for representing detailed geometry distributed over a large volume in a single coordinate system. Different branches within the quadtrees are assigned to different local coordinate systems, which are centered such that precision is maximized, and oriented to locally preserve natural directions such as "up", which can be exploited by the terrain geometry level of detail algorithm.

VGIS has been designed to be portable across a spectrum of different platforms, but is primarily targeted towards high end graphics architectures such as Silicon Graphics workstations. Certain platform specific extensions are optionally incorporated at compile time to maximize the system performance. The system is layered on top of portable standardized libraries such as OpenGL and POSIX threads, and provides a level of indirection for interfacing with the windowing subsystem, e.g. X11/Motif for Unix platforms.

4 SYSTEM OVERVIEW

The VGIS system consists of a dataset pre-processing component, as well as the actual run-time visualization display system. Optionally, external processes may provide real-time or remote data input. Due to the vast volumes of data that make up the global terrain database, the data is prepared off-line and structured in a form that closely matches the internal representation so that it can be paged into main memory efficiently. This form of pre-processing includes data format conversion, reprojection, resampling, data analysis, and data synthesis. In order to maximize the run-time performance,

non-interactive processes such as simulation, real-time data acquisition, and remote queries can be separated from the run-time system and be executed as remote processes. A number of server threads run in the back end of the run-time system that communicate with the external processes and fetch data from local disk. In addition, these threads perform intermediate tasks such as data transformations (e.g. conversion of height field data to Cartesian coordinates), data synthesis and initialization (e.g. generation of LOD parameters, synthesis of different image types), and repackaging of the data into the internal representation whenever necessary (e.g. parsing of DIS packets and GIS queries). Such processing is sometimes necessary to limit the amount of data stored on disk, and also allows VGIS to interface with external modules such as ARC/INFO GIS servers and ModSAF DIS simulators.

The servers are broken down by data type (e.g. terrain, symbology, GIS) and are run as independent threads. Each server handles requests from a number of clients that manage the corresponding data type. As mentioned in the previous section, VGIS supports the concept of multiple, independent views, each corresponding to a window on the screen. For example, one view could display the user riding a moving vehicle in three dimensions, another may be a "God's eye view" looking down upon the vehicle, while a third view could be a 2D overview map of a larger area. For each view, there is a *view module* that contains data managers for each data type. For example, there is a terrain manager that makes data requests to the terrain server and handles level of detail management, surface queries, terrain rendering, etc. Similarly, there is an object manager that manages animation and display of 3D symbology and protrusive features. The actual on-screen rendering is done by a single dedicated render thread. This thread runs asynchronously from all other threads to provide maximum rendering performance and interactivity. Within each view module, the scenes that are to be rendered are prepared and buffered in data structures similar to OpenGL display lists. These display lists are then sent to the render thread for display. The display lists are typically updated less frequently than the scenes are rendered, thus allowing the renderer to reuse a display list over several frames.

To minimize the response time, the renderer fetches copies of the most recent view parameters (i.e. position, orientation, field of view, etc.) for each view at the beginning of each frame. The view parameters are updated by a single user interface (UI) thread, which also acts as an overall manager of the system. Whenever a UI event is generated, e.g. from an input device or from remote commands, the corresponding view parameters are updated for that view.

An overview of the VGIS architecture is given by Figure 1. The following sections describe each of the system components in more detail.

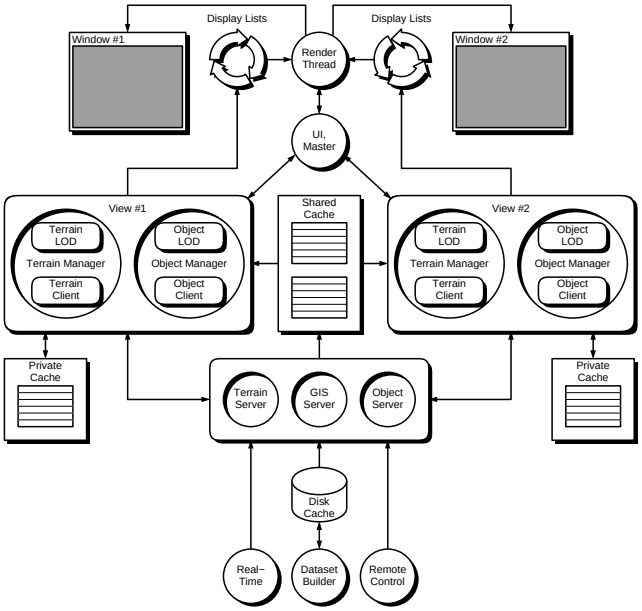

Figure 1: Overview of the VGIS system architecture. This figure illustrates two independent views but can be generalized to an arbitrary number. Modules are represented by rounded boxes, processes and threads by circles, data structures by rectangular boxes, and data flow and communication by arrows.

4.1 Pre-Processing

Data processing in the VGIS system comes in three forms: off-line pre-processing, which is done once per dataset; intermediate, on-line processing, which is performed during the data paging stage; and real-time, on-line processing, which is typically done once per frame. The purpose of the pre-processor, or *dataset builder*, is to gather the different types of data and transform them into a format that is readable by and quickly accessible to the run-time system. These tasks are inherently compute intensive and cannot be performed by the run-time system at high enough rates. Additionally, the pre-processing only has to be done once for a given dataset, so it makes sense to pay the penalty of assembling a dataset up front.

While the pre-processor may handle a large variety of different data types, we will limit this discussion to the processing of static terrain data. Some of the basic data types within this domain include terrain geometry (e.g. "raw" elevation, elevation corrected for non-protrusive surface features such as roads and rivers, as well as the surface features themselves), imagery (e.g. phototexture, Arc Digitized Raster Graphics—or ADRG, pre-shaded relief maps, contour maps, etc.), and surface properties (e.g. surface roughness parameters, bounding volumes). The source data may come in a variety of file formats, which the pre-processor must translate to a single common format. In addition, for each data type, the source data may consist of multiple, variable resolution, possibly overlapping/nested datasets. The pre-processor has the ability to layer these and composite them into a single dataset if so desired.

As mentioned above, VGIS utilizes a quadtree structure for organizing multi-resolution terrain data in a hierarchical manner. Each node in a quadtree identifies a fixed, square area (in geodetic lat/lon coordinates) at a given discrete resolution. Additional constraints force the dimensions of the raster tiles associated with the nodes to be powers of two,[1] and the post spacing is successively doubled on each consecutive level up the tree. To facilitate the use of this data structure, each terrain dataset must be resampled to one of the discrete pre-determined resolutions. To accommodate fast paging, the data is also resampled and generated for all the internal nodes of each quadtree, resulting in what is commonly referred to as a *pyramid structure*.

The pre-processing of the data is done entirely in geodetic coordinates, assuming a pre-determined geodetic datum.[2] This ensures good registration between different source datasets and avoids the discontinuity problems often associated with flat-projected data. Both the height field and imagery are represented as regular grids. Hence, we do not require additional compute time to triangulate the height field—the different discrete levels of detail of the geometry are rather represented by the pyramidal regular grid structure, and further refinement is employed on-the-fly by the run-time system. See [20] for a detailed discussion of the height field level of detail algorithm. To further improve dataset registration, *alpha chan-*

[1] The elevation height field nodes have raster dimensions $2^n + 1$ as their boundary rows and columns overlap.

[2] In the current implementation, the WGS-84 datum is used.

nels can be added to the source data (both imagery and elevation) to allow blending/smoothing across dataset boundaries as well as masking of the data, e.g. for irregularly (non-rectangular) shaped datasets or for regions where data is missing.

The dataset builder has been designed to be very flexible and extensible in terms of incorporating many different data types, and is structured in a modularized manner to allow for fine-grained parallel execution. The different data types are declared by the user using a class hierarchy language. For each class, a number of methods/modules are defined that are used for reading, writing, and processing the data, with bindings to implementations of the methods. Class inheritance can be exploited to avoid redundant methods and class members. The modules are then connected in a data flow network similar to the structure provided by visualization systems such as *AVS* and *Data Explorer*, that is the output of one module is piped to the input of another set of modules. A set of operators are provided for expressing which modules are allowed to run in parallel, and which parts must execute in series. The output data must be classified into a few number of meta-types, such as geometry and imagery, that the run-time system understands how to manage and display. A repository of generic and specialized modules is maintained which can be supplemented to handle near arbitrary data types, both by the pre-processor and the run-time system. The dataset building process is entirely automated, making it very easy to insert and remove data. This is important for facilitating real-time acquisition and integration of data such as up-to-date satellite imagery.

4.2 Run-Time System

The run-time part of VGIS has been designed to support highly interactive frame rates. As such, it relies heavily on a multithreaded, fine-grained task distribution. Since the display update rate is often of higher importance than the scene update rate, including level of detail selection and animation, more resources are allocated towards satisfying a minimum frame rate. In [20], we propose a terrain geometry level of detail algorithm that generates "continuous" levels of detail on-the-fly. While being highly efficient in selecting the vertices and triangles that make up the terrain surface for a given view, the recursive traversal of the terrain data structures for triangle stripping is a bottleneck that limits the rendering rate to approximately 10 frames per second for pixel-accurate full-screen views. As a lot of frame-to-frame coherence is evident in the resulting triangle strips, it is often satisfactory to perform the LOD management less frequently, say 1–5 times per second, and trade the scene update rate for higher display rates. We accomplish this by decoupling the two tasks and separate them into two different, asynchronous threads. This scheme allows the scene to be redisplayed (with potential changes to the view between frames) until new parts of the scene have been generated and submitted by a scene manager. In addition, we further segregate different data types, recognizing that different data products may require different display update rates. For example, animated vehicles may be updated at a higher rate than the terrain geometry.

By and large, our approach has been to identify the major bottlenecks, such as blocking system calls, and isolate them from the time-critical components. This means offloading the I/O intensive, high latency data paging from the scene managers, and feeding the render thread with "ready to render" display lists. The resulting architecture forms a hierarchy of successively lower priority tasks. Each of these tasks is discussed in the following sections.

4.2.1 Paging and Intermediate Processing

The server layer in VGIS provides an interface to accessing external data and transforming it into the internal data structures employed by the display subsystem. Since the latencies involved in disk transfers and inter-thread communication are significant compared to display update times, the servers are decoupled and run asynchronously from the more time-critical pieces of VGIS. However, since the requirements on bandwidth between servers and clients are quite high, we chose to integrate the servers with the back end of VGIS and have them communicate with the clients via shared memory. In fact, the servers have direct access to the internal caches of VGIS, limiting the communication to small request messages and acknowledgements (see Figure 1). For less time-critical services (e.g. GIS queries), another (external) layer of servers can be built on top of VGIS. An example of such an external server is the ModSAF battlefield simulator which sends DIS packets over a Unix socket (possibly from another machine) to the internal object server, which in turn interprets the packets and reformats them for internal storage. Thus, the VGIS servers have two tasks: "data paging" and "intermediate processing," or preparation and initialization of the paged data. In this section, we will focus on the most bandwidth-consuming service; terrain paging. Other servers operate in a similar manner but under different conditions, and are typically not limited by bandwidth.

For each view, there is a terrain manager thread, part of which is a client module. The client module is responsible for making data requests to the terrain server whenever data of some type and resolution is needed for a particular area, and taking the appropriate actions upon notification by the terrain server that the request has been serviced. When data is needed for a node in a quadtree, the client allocates space for the data within a shared cache and sends a message via a shared memory priority queue to the server. Message priorities in this queue are changed dynamically according to the importance of the associated request as determined by the level of detail manager. Thus, requests that gradually become less important, or even obsolete, sift towards the end of the queue and get serviced only when no higher priority requests remain in the queue. This is important as the paging rate, during short bursts of requests, is typically much lower than the request rate. The server dequeues the highest priority request and either reads the data from disk if it exists, or synthesizes the data from other sources (or possibly a combination of both). After transferring the data from disk, the server may have to do additional processing. In the case of elevation data, the server reads a height field raster from disk, and then proceeds to transform the height field lat/lon/height coordinates into an array of Cartesian vertices. LOD state and other parameters are also generated and initialized before the request is completed, and the terrain client making the request is notified by returning an acknowledgement message.

Modules can be attached to the servers to handle paging of "arbitrary" data types. A "read module" for each data subtype must be registered with its corresponding server, and the core of the server is merely a dispatcher of jobs to be executed by these modules. These independent modules can be made to run in parallel to further increase the throughput. In this fashion, a number of data types can be synthesized from existing data and be easily integrated with VGIS without having to restructure the server. For example, a module can be added that synthesizes contour map images on-the-fly from existing elevation data. In this particular case, the elevation data may not even have to be read in if it already resides in memory since the servers have access to the shared memory cache.

4.2.2 Internal Representation and Caching

In the previous sections, we briefly mentioned the quadtree and shared cache data structures. These two structures constitute the basic components of the internal terrain representation. Rather than having a single quadtree represent the globe, we chose to subdivide

the globe into a small number of quadrants.[3] Each such quadrant is further subdivided and organized by a hierarchical quadtree structure. A node in a quadtree corresponds to a raster tile of fixed dimensions and lat/lon resolution according to the level on which it appears in the quadtree. Quadnodes are identified by "quadcodes," which are constructed in a manner similar to the indices of array representations of binary trees, that is the children of a node with quadcode q are identified by $4q+1$ through $4q+4$. In addition, the quadcode contains a quadtree identifier which allows each quadcode to uniquely identify an area on the globe.

In order to conserve memory, the static, view-independent data associated with a node is stored in a shared cache. This allows multiple terrain managers to access the same data without having to replicate it. The shared cache itself is implemented as a set of hash tables, one for each data type (e.g. elevation, phototexture, ADRG), which have enough slots to hold all the quadnodes that exist in the dataset. These slots are initially empty, and are filled with terrain data whenever a request is processed by the terrain server. If a node is no longer needed by any of the terrain managers, the space for it is deallocated. The quadcodes are used as hash keys for accessing nodes in the hash table. Since the hash table slots are initialized at startup, the terrain managers know what nodes exist externally such that no invalid data requests are made to the server. To avoid race conditions on the hash tables, each table is supplemented with a lock or a semaphore. Note that the hash tables need to be locked only on transitions such as when space for a node needs to be allocated or freed (ensuring that multiple terrain managers do not allocate or free a node's data simultaneously), or whenever a manager begins or ceases to reference a node. Reference counts are maintained so that the managers know when to allocate and free nodes.

The actual quadtree structure and any view-dependent parameters—such as vertex LOD state information—are stored in a view-private cache (Figure 1). The quadtree data structure is mostly a skeleton that indicates the presence of nodes, while the shared cache holds most of the actual terrain data. This scheme can be extended to data other than terrain, e.g. for moving vehicles and stationary objects, where pointers are used to reference the appropriate level of detail model within the shared cache.

In addition to spatially organizing the data, the quadtrees also define the boundaries of the aforementioned local coordinate systems. If a single, geocentric coordinate system were used, and assuming 32-bit single precision floating point is used to describe geometrical objects,[4] the highest attainable accuracy on the surface of the Earth is half a meter. Clearly, this is not sufficient to distinguish features with details as small as a few centimeters, e.g. the treads on a tank. As a matter of fact, many of the terrain datasets that have been used in VGIS have 10 centimeter resolution. This lack in precision results in "wobbling" as the vertices of the geometry are snapped to discrete positions. To overcome this problem, we define a number of local coordinate systems over the globe, which have their origins displaced to the (oblate) spheroid surface that defines the Earth sea-level. The origins of the top-level coordinate systems are placed at the geographic centers (i.e. the mean of the boundary longitudes and latitudes) of the quadtree roots. While the centroid of the terrain surface within a given quadrant would result in a better choice of origin in terms of average precision, we decided for simplicity to opt for the geographic center, noting that the two are very close in most cases. The z axis of each coordinate system is defined as the outward normal of the surface at the origin, while the y axis is parallel to the intersection of the tangent plane at the origin and the plane described by the North and South poles and the origin, that is the y axis is orthogonal to the z axis and points due North. The x axis is simply the cross product of the y and z axes, i.e. $\hat{x} = \hat{y} \times \hat{z}$, and the three axes form an orthonormal basis. This choice of orientation is very natural as it allows us to approximate the "up" vector by the local z axis, which further lets us treat the height field as a flat-projected surface with little error. Hence, the height field LOD algorithm, which is based on vertical error in the triangulation, does not have to be modified significantly to take the curvature of the Earth into account. The delta values (see [20]) are however computed in Cartesian rather than geodetic coordinates to avoid over-simplification of "flat" areas such as oceans. Figure 2 illustrates the local coordinate systems for a few quadrants.

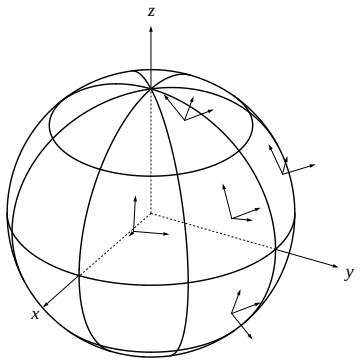

Figure 2: Local coordinate systems for the quadtree roots. The labeled axes correspond to the conventional Earth Centered, Earth Fixed Cartesian XYZ global coordinate system.

Using the above scheme, the resulting worst case precision for a $45^\circ \times 45^\circ$ quadrant is 25 cm—not significantly better than the geocentric case. We could optionally use a finer subdivision with a larger number of quadrants to obtain the required precision. However, this would result in a larger number of quadtrees, which is undesirable since the lowest resolution data that can be displayed is defined by the areal extent of the quadtree roots. Hence, too much data would be needed to display the lowest resolution version of the globe. Instead, we define additional coordinate systems within each quadtree. In the current implementation, we have added 256×256 coordinate systems within each quadtree, resulting in a 1 mm worst case precision.

4.2.3 Terrain Level of Detail

VGIS applies level of detail techniques to simplify both geometry and texture detail. This is necessary to maintain interactive frame rates as the global views typically contain millions, or even billions, of surface polygons, with gigabytes worth of imagery. Both of these techniques guarantee to meet an upper bound on the screen space simplification error, and the error bounds can be manipulated interactively by the user until sufficiently high frame rates are obtained.

The terrain geometry LOD algorithm is presented in [20], but we will give a brief overview of it here. The algorithm proceeds in two stages; a coarse-grained simplification in which quadnodes are selected at the appropriate resolution, followed by a fine-grained simplification in which individual vertices within each node are decimated. The decision as to whether a vertex can be removed is based on the screen space distance a vertex travels from its original position to the resulting surface if it were to be removed. The corresponding world space distance is referred to as the vertex's "delta value". If this distance is smaller than a screen space threshold, the vertex is decimated and further simplification of nearby vertices is considered recursively. The first stage—coarse-grained

[3] In the current implementation, thirty-two $45^\circ \times 45^\circ$ quadrants are used.

[4] This is typically the highest precision available in current graphics hardware.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.