Published in the Proceedings of the INTERNATIONAL CONFERENCE ON DISCRETE GLOBAL GRIDS (2000)

Real-Time Global Data Model for the Digital Earth

Nickolas Faust[1,2], William Ribarsky[2], T.Y Jiang[2], and Tony Wasilewski[1]
[1]GIS Center and [2]GVU Center
Georgia Institute of Technology

ABSTRACT

A global data model is presented for the digital earth that provides scalability and fast access. It is shown how the same global hierarchical structure can be used for a wide variety, and perhaps all types, of geospatial data. To maximize efficiency, particular types of data can, at prescribed levels of detail, transition from the global model to type-specific data structures and detail management schemes. This framework has been applied successfully to a variety of data including terrain elevations, phototextures and imagery, maps, buildings, moving or flying vehicles, weather, and other data. Further, the framework provides a geospatial visual data mining approach where one can navigate continuously from global overviews to high resolution local views. This paper presents results for several applications.

Keywords: visualization, global terrain, data model, digital earth, decision-support, large scale data, GIS

1 INTRODUCTION

To attack the problem of dealing with increasingly vast stores of global information for the digital earth, we describe in this paper a general approach to data organization and real-time exploration based on a novel global hierarchical data model. Our recent work has revealed that this framework can be quite generally applied to the earth and anything on it, above it, or even below it. This includes terrain elevations, phototextures and imagery, maps, buildings, moving or flying vehicles, weather, and other data. Further, the framework provides a geospatial visual data mining approach where one can navigate continuously from global overviews to high resolution local views. Because it has an efficient data organization and paging capability closely coupled to a view-dependent data requesting mechanism (that is, it is adjustable based on the requirements of the display platform), the framework is also quite flexible and has been applied to a range of single and networked systems ranging from a single-processor PC to immersive systems with multiple projection screens and coupled computers. In this paper we will discuss our geospatial model, how it fits into an overall framework, and how we have applied it to different types of data in different environments. We will also show that the model is applicable to distributed and Internet-based systems and will discuss our work in that direction.

GVU Center, Georgia Tech, Atlanta, GA 30332-0280
Email: {nick.faust, tony.wasilewski}@gtri.gatech.edu
{ribarsky, jiangf}@cc.gatech.edu

For the following reasons our approach offers a general framework for digital earth applications:

- It accepts and integrates all types of geospatial data into a global framework.
- It is scalable to very large data stores and to distributed environments.
- It provides an interactive visualization framework.
- It supports discovery of new information via navigation in context with unfolding detail.

We have implemented these capabilities in a system called VGIS [Lin96, Lin97].

Our approach starts with a hierarchical structure for optimal interactivity for data exploration. We use a novel "forest of quadtrees" with nested coordinate systems for handling global scale terrain data and permitting timely paging of collections of objects in support of real-time navigation. We have found that one can effectively employ this hierarchical structure for a wide range of geospatial data as long as methods adapted to the specific type of data (e.g., terrain versus buildings) are applied at the lower (detailed) levels.

An important aspect of our approach is the idea of a "view" and "view-dependence". We impose a viewpoint and a limited viewing window on the data presentation. In general there are things within the view and outside the view, things that are close or far away, and things that may be obscured because they are blocked by other things. By view-dependence, we mean that we organize the retrieval of data by these view-dependent factors. Thus we might only retrieve the most prominent details for data that are far away and might not retrieve at all data that are out of view. Of course this approach is useful and efficient for visualization, but it is also a powerful organizing principle for all data handling (for example, GIS data queries). Even for these operations there is an implied viewpoint and viewing window. This approach also supports the ideas of navigation and context. Thus we access details in extended datasets by navigating the data, and we do this while seeing surrounding contextual information that isn't our main focus but can still provide important supporting details.

In this paper, we present these ideas in detail. We also describe how large collections of objects, such as buildings and weather clouds, can efficiently fit, along with terrain, into the geospatial hierarchy. Application results demonstrate that our approach is both scalable and general because it is able to handle both large scale global terrain information and multiple collections of objects (e.g., cities and weather data) placed around the earth with full interactivity and without extensive memory load. Finally, our approach shows that levels of detail can be naturally incorporated to provide improved detail management.

2 RELATED WORK

There has been a large body of previous work that has considered how to organize and visualize geospatial data. We will not attempt to cover all this work here but rather will focus on the most relevant research and applications. Representing data that span the entire globe requires special data structures that efficiently account for the earth's curvature. Most of these are based on quadtrees [Sam84]. Fekete has developed spherical quadtrees [Fek90] as a global representation for the earth. The main difference between Fekete's method and ours is that he tesselates the earth's surface with a a subdivided icosahedron, while VGIS uses "square" geodetic patches, whose geometry conform to the lat/lon representation commonly used for terrain elevation and imagery datasets. Similar hierarchical data structures have been shown to be useful for global GIS [Goo92].

Much work on the tesselation of terrain 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 [Gar95, Sch94]. In addition hierarchical triangulation representations have been proposed that lend themselves to multiresolution level of detail (LOD) algorithms [DeB95, DeF95]. Regular grid tesselations, which is the form used in our approach, have also been implemented as terrain approximations [Cos90, Duc97]. Although recent work shows that improved data compression can be attained with TINs [Kim99], we feel that there remains a significant advantage in using regular grids in both dataset compactness and efficiency of dataset building. (See Sec. 4 and [Lin97])

VGIS treats the terrain as a single, connected surface for rendering using a "continuous, view-dependent" LOD representation. Similar approaches are described in in [Cos90, Duc97, Hop98, Xia96]. However, VGIS is the first system to support support real-time view-dependent, pixel-accurate multiresolution rendering of high resolution global datasets.

A number of visualization systems have been implemented that integrate 3D visualization techniques with large 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 [Nis93]. Other systems emphasize tight integration of the 3D visualization with the powerful spatial analysis capabilities of GIS [Erv93]. VGIS enables real-time, highly interactive 3D visualizations of the spatial data by building these capabilities into the system from the bottom up.

Recently there has been work that addresses interactive visualization of very large, out-of-core datasets in applications from 3D flow visualization to large scale terrain visualization. From what has been done so far, it is clear that application-control and domain-dependent data organization are essential to achieving good performance [Uen97, Cox97]. Relying on system virtual memory, for example, frequently results in thrashing and abysmal performance. Often a spatial and hierarchical partition of the dataset is used so that one can efficiently load only needed segments. All of these out-of-core techniques consider the paging of continuous volumetric or terrain data. VGIS fits with these application-controlled out-of-core techniques but, unlike the others, it considers more diverse data such as terrain, time-dependent volumetric data, and collections of discrete 3D objects.

3 AN EFFICIENT GLOBAL HIERARCHICAL STRUCTURE

We have built a quadtree and shared cache data structure that together constitute the basic components of our global data model. The global structure is divided into 32 zones, each $45^{o} \times 45^{o}$ [Dav98, Lin97]. (See Fig. 1.) Each zone has its own quadtree; all are linked so that objects or terrain crossing quadrant boundaries can be rendered efficiently. We chose the number and extent of zones based on empirical observations of memory requirements, paging overhead, geometric accuracy, etc. 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 built in a manner similar to the indices of 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 that allows each quadcode to uniquely identify an area on the globe. This structure is replicated in the underlying disk management system so that files are aligned with the quadnodes in the set of linked quadtrees

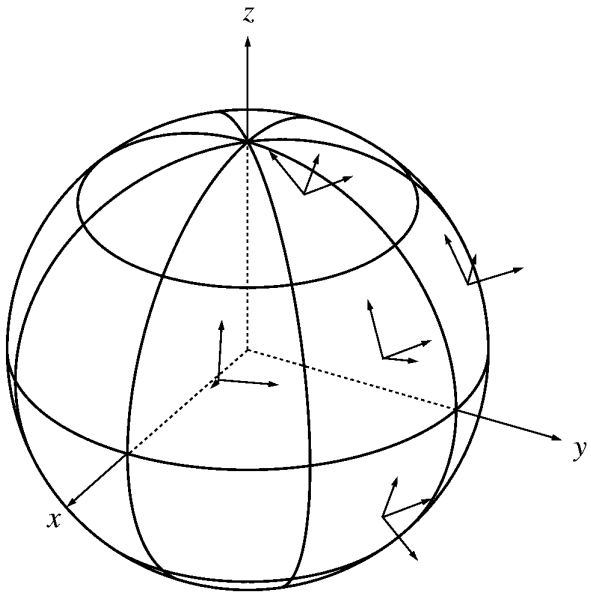


Fig. 1 The Earth divided into 32 zones.. The labeled axes correspond to Earth Centered, Earth Fixed Cartesian XYZ global coordinate systems for each zone.

The quadtrees also define the boundaries of local coordinate systems. If a single, geocentric coordinate system were used, assuming 32-bit single precision floating point is used to describe object geometries, 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, some 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, which is present in other large scale terrain systems such as T_Vision [Gru95]. We have developed a new approach to overcome this problem [Lin97]; 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 zone 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 locally points due North. The *x* axis is simply the cross product of the *y* and *z* axes, 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. However, the delta values (see [Lin96]) must be computed in Cartesian rather than geodetic coordinates to avoid over-simplification of constant-elevation but curved areas such as oceans. Fig. 1 shows the local coordinate systems for a few zones.

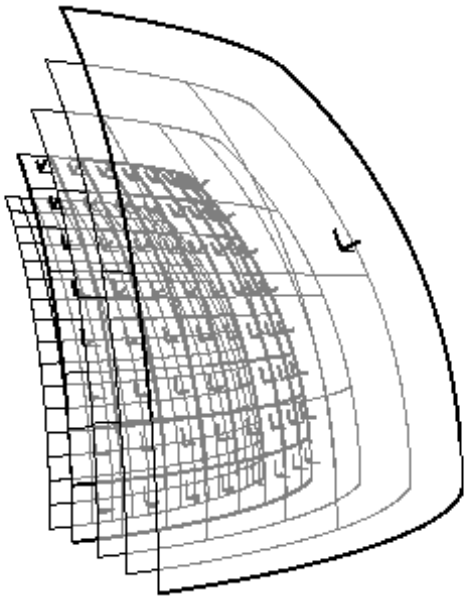


Fig. 2 Nested coordinate systems in a quadtree. 8 x 8 smaller coordinate systems appear 3 levels below the root node.

Using the above scheme, the resulting worst case precision for a 45^{o} x 45^{o} zone is 25 cm---not significantly better than the geocentric case. We could optionally use a finer subdivision with a larger number of zones 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 x 256 coordinate systems within each quadtree---one coordinate system per node, eight levels below each root node---resulting in a 1 mm worst case precision. Fig. 2 illustrates a subset of these nested coordinate systems. The terrain and object managers keep track of which coordinate system to use among these thousands of systems and can even transition between coordinate systems for extended objects.

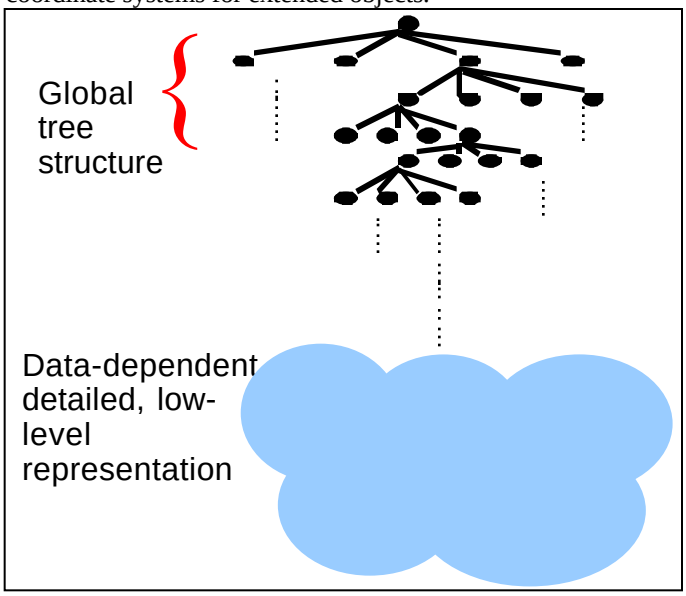


Fig. 3 General global hierarchical framework.

The general approach to using the hierarchical structure is illustrated in Fig. 3. In each zone the quadtree is traversed to a certain level, depending on the type of geospatial data. Below this level a non-quadtree detail management scheme is used that depends on the detailed characteristics of the data. Thus, for example, buildings and terrain have different levels at which separate non-quadtree detail management schemes take over. We describe these schemes further below.

This global hierarchical, nested structure will handle the earth and everything on it at levels of detail from global overviews to fine resolution close-ups. Navigation between these extremes involves changes of up to 10 or more orders of magnitude. We are now extending the structure to everything over the earth including weather and other atmospheric effects [Rib00]. Since the atmosphere is only a thin layer in proportion to the extent of the earth's surface, this should be an efficient approach.

Caching. To conserve memory and promote efficiency, the static, view-dependent data associated with active nodes of the hierarchy are stored in a shared cache. This allows

multiple managers for the various data types to access the data without having to replicate it. The shared cache consists of a set of hash tables, one for each data type (e.g., elevations, phototextures, feature data, buildings, moving objects, etc.), which have enough slots to hold all the quadnodes in the dataset. These slots are initially empty and filled with geospatial data whenever a request is processed by a particular server. If a node is no longer needed by any of the 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 managers know what nodes exist externally such that no invalid data requests are made to the server. At present all quadtree hash tables are loaded at startup. In the future to maintain scalability, we will impose a structure where only the high level quadtree tables are loaded at startup with an additional paging and caching mechanism to bring in more detailed portions of the quadtrees as they are needed.

Paging. To support parallelism and expandability, there are separate paging threads for the different types of geospatial data. Each thread has a server and a manager. The server loads pages from disk while the manager decides which cells should be loaded (taking into account such things as user viewpoint and navigational speed) and passes it along for display or analysis. This communication path supports a demand-paging approach such as that of Cox and Ellsworth [Cox97]. When data are needed for a node in the quadtree, the manager allocates space in the above shared cache and sends a message to the manager. Message priorities in this queue are changed dynamically according to the importance of the associated request as determined by the manager. Thus, requests that gradually become less important sift towards the end of the queue and get serviced only when no higher priority requests remain in the queue.

We have found that the above page priority procedure sometimes falls short when handling global data. Users of such data frequently fly quickly from a global view where the terrain elevation and imagery data are at 8 Km resolution to views close to the ground where the data are at 1 M resolution or higher, and there may be hundreds or more buildings in view. If the user flies in too fast, the traversal of linked quadtrees by the terrain manager falls well behind the user's navigation. The process can stall in this case, and the pages for the scene currently in view can take quite long to arrive.

Unfortunately the system cannot just jump to the appropriate position in the quadtree. The quadtree has to be traversed to get important properties information, especially quadcell linking data but also geospatial bounding boxes and other data, that are necessary to determine if the object or other data should be displayed or not. To address this problem we created separate sets of skeleton trees, one set for each geospatial data type [Dav98, Dav99]. (See Fig. 4.) This separate structure provides properties information but is lightweight so it can be traversed quickly. Large segments of the indexing trees reside in main memory for fast access. With the flexibility of this scheme we can skip one or more levels before paging in object data. A predictive mechanism is instituted based on user navigational speed and viewing direction to help predict where the terrain manager should skip.

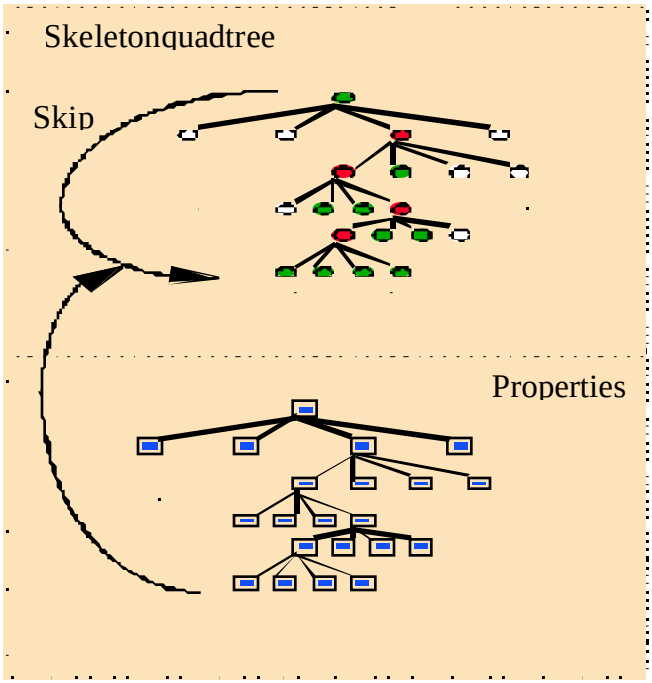


Fig. 4 Skeleton quadtree structure with ability to skip quickly between nodes. If a node does not contain data, properties are retrieved, which tell whether data are needed for the current view and where the data are located.

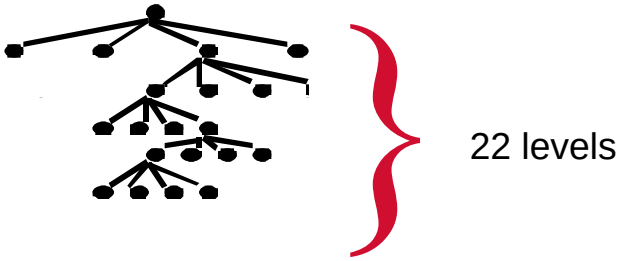
Since the managers are receiving continuous updates from the user via the user interface, they can use these in their requests to the servers. For example, the object manager can expend more detail on buildings or other objects in the center of the scene.

4 TERRAIN AND STATIC OBJECT DETAIL MANAGEMENT

We give examples of the type-specific detail management schemes illustrated in Fig. 3. We concentrate on terrain and static objects, but the framework can be extended to other types of data.

Terrain. We start with a brief overview of terrain detail management, described more fully in [Lin96] and [Lin97], and present some new capabilities. After the upper level traversal of the quadtree, the algorithm arrives at the leaf nodes. For high resolution data at, say, 1 M spacing this is typically after about 22 levels. The data in these nodes are arranged in a regular grid with a height value at each vertex. (See Fig. 5.) Using the latest viewpoint and view direction, the algorithm determines on-the-fly which vertices to include in the scene display. The decision as to whether a vertex can be removed is based on the screen space distance the vertex travels from its original position to the resulting surface if it were to be removed (i.e., if it's height component were flattened out). 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. After the final set of vertices have been selected, the terrain manager produces a single triangle strip for each quadnode, which is simply a list of alternating vertices and texture coordinates enclosed by begin/end triangle strip commands. These display lists are a new capability that emulates the OpenGL counterpart, but are not the same. They are used because the scene managers cannot employ OpenGL display lists directly as only one thread at a time is allowed to issue graphics commands within an OpenGL context. Fig. 6 shows a terrain surface tessellation resulting from application of the continuous LOD algorithm. The threshold in this view is 1 pixel. Note that shading has been applied to the geometry before simplification to retain detail. Even for rough terrain at 1M resolution, this procedure can reduce the number of triangles displayed by a factor of 100 or more [Lin96]. If the terrain is smooth, the reduction is even greater. Further, with a 1 pixel threshold the visual quality of the scene displayed is not perceptually different from a scene rendered with all the triangles.



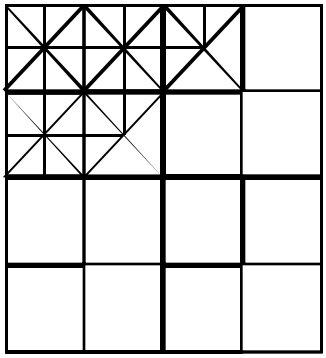


Fig. 5 Quadtree structure specialized for terrain elevations. Each vertex has a height value. For a given view, vertices are removed until a specified error in the height field is reached.

In addition to terrain elevations, we must manage terrain textures. These can be phototexture images, maps, textured shapes describing vegetation patterns, and so on. The texture LOD management bears some similarities to the geometry simplification. Rather than projecting the largest delta value to the screen, the largest (in screen space) *texel* is projected and compared to a threshold. The largest texel is found after taking into account the viewpoint-to-texel distance and the angle the texel normal makes with the viewpoint-to-texel vector. We here assume that all texels lie in a horizontal plane, but may be distributed in elevation within the bounding box of a node. The bounding box is determined by the area of the terrain patch for that node and the maximum and minimum height values within the patch. As a result, low detail is used when the terrain surface is viewed from the side, while relatively higher detail is required for top-down views.

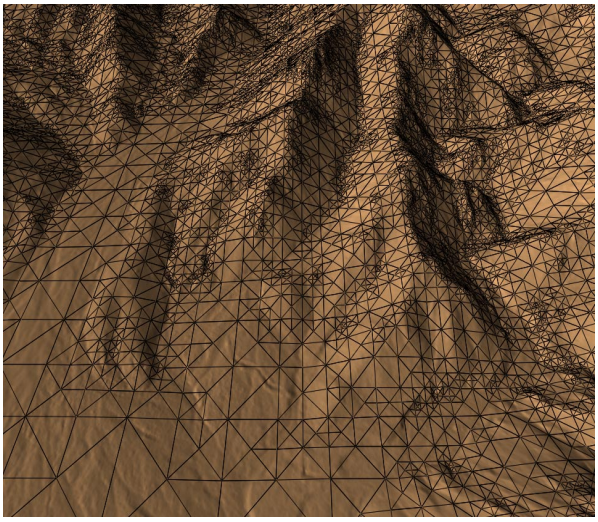


Fig. 6 Terrain surface tesselation showing dynamic detail management based on viewpoint and on terrain roughness.

Static Objects. If we have several cities in our global dataset, we want to quickly determine that the viewer is navigating towards, say, Los Angeles and does not need detailed data for Bombay, Houston, or even San Francisco. Further, as the user navigates to a certain neighborhood within a city, we don't want to page in and then have to cull, one-by-one, buildings for a neighborhood on the other side of town.

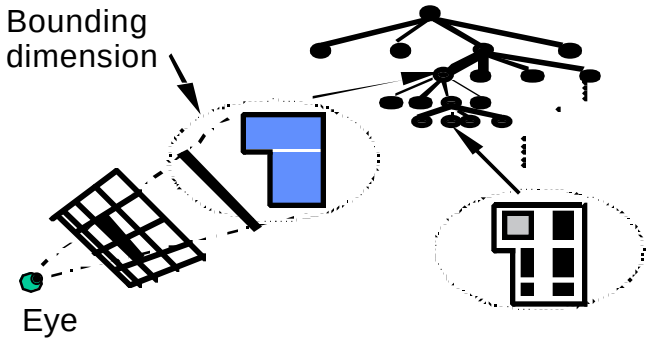


Fig. 7 Viewpoint-dependent detail management for static objects. The bounding dimension of a lower detail representation is checked. If necessary, the higher detail representation (lower right) is used instead.

To achieve these goals we use the global hierarchical structure with a few additions. Object pointers are loaded at the quadnodes determined as described above. Accompanying the pointer is the object location and a bounding dimension obtained from the largest dimension of the object bounding box. Using the viewpoint, object location, and bounding dimension, the system finds a maximum size for the object in screen space in terms of pixels. (See Fig. 7.)

When considering small buildings at urban densities, we use a more automated and compact linked cell mechanism. We must first determine the level to convert from the quadtree to a specific detail management scheme appropriate for buildings or other static objects. Let us assume a quadcell of side L_q. (The bounding box and bounding dimension in Fig. 7 would be of order L_q.) Now let's assume an object with maximum dimension L_o. If $L_o < L_q$, then only the 8 surrounding quadcells might contain objects that would extend into a central quadcell. In fact if we divide the central cell into 4 quadrants and further assume that all objects are placed in the quadcells that contain their centers, the maximum number of quadcells whose objects could overlap this quadrant would be 4, the central cell plus the 3 nearest quadcells to the quadrant. Our approach is to go as deeply as possible using the very efficient quadtree but not to permit more than 4 neighboring (or "linked") cells for considering overlapping objects, since there will be increased overhead from keeping track of the links and from having to consider all the objects in the linked cells. The collection of objects in the linked cells would have to be considered in view frustum culling, collision detection, and many other operations. Thus we choose cells for which $L_o < L_q$. For buildings in an urban setting, L_o could be about 50 meters, the dimension of a typical city block. If a typical building is the size of a house, say 10 meters on a side, we would have to consider on average no more than 100 or so buildings for 4 linked cells. Note that occasionally we will have to consider linked cells that could be in more than one quadtree.

We must also consider carefully extended objects such as a stadium, a large factory, or very large objects created via detail management. These might require considering several blocks as one object. If we use the largest such object in the database to determine the leaf node level of the quadtree, we might also end up with cells containing several hundred smaller objects. To obviate this problem we have discrete representations of such large objects at successive levels of the quadtree, each representation carrying its own list of linked cells. Thus if we flew from outer space down towards an urban area, we might first see a phototextured shape representing the downtown area, which would then be replaced by more detailed shapes representing collections of blocks and tall landmark buildings, and finally these would be replaced by shapes for individual buildings. Although the present system switches between discrete representations, one can imagine a more sophisticated process with more continuous switching of detail.

5 RESULTS

The global data model and the VGIS visualization interface use standard interface and programming elements and run on a variety of platforms. These include UNIX platforms, such as SGI and Sun, and PCs running Windows NT. A variety of display configurations are used including desktop, virtual workbench, and NAVE. (For more details see [Res00].) The virtual workbench and NAVE both use large projected displays that present stereoscopic scenes. The NAVE is a low-cost version--created at Georgia Tech--of the immersive, multiple screen CAVE. It runs entirely on PCs, rather than the SGIs used for the CAVE. Desktop versions of VGIS use the standard mouse interface, but the projected systems, especially the virtual workbench, use head and two-handed tracking. The NAVE also uses a joystick interface.

The results we present are from a single, global database of several gigabytes. Our Army colleagues have built VGIS databases of well over 20 GB without loss of capability. As stated above, we expect the model to scale to much larger data and will be testing this scaling soon. All scenes shown (Figs. 8-10) can be reached by continuous navigation from one place to another. Since typical databases are not likely to have terrain data everywhere at high resolution, our geospatial framework permits accurate placing of high resolution insets on lower resolution backgrounds and also nesting of several resolutions. For example, the state of Georgia (at 60 M resolution) is placed on the U.S. (at 1 Km resolution), and within Georgia are several nested databases around Atlanta (greater Atlanta at 10 M resolution, central downtown at 1 M resolution, and Georgia Tech at 1 foot resolution).

It is important to have the capability to build such large databases incrementally. Unlike most systems with real time data access, the VGIS database can be built incrementally with an efficient terrain dataset builder tool [Lin97] that inserts terrain data in a time that scales with the size of the inserted piece. This time will tend to become much smaller than the time to rebuild the entire database as the latter grows very large.

For navigation of the global database, typical frame update rates are 30 frames per second (fps) or more for an overview of a continent and remain at or above 15 fps even for flying close to high resolution terrain. These frame rates are for an SGI Infinite Reality Engine. Rates on a 300 MHz PC with TNT2 3D graphics card are slightly lower. When a large number of buildings are in view, the frame rates are somewhat lower on either platform. The paging delay for high resolution terrain depends on the speed at which one approaches an area. If one flies from a global overview to a close-up view, high resolution terrain elevations and imagery may be delayed up to a second or two on the Reality Engine and 3-4 times longer on the PC. More gradual fly-ins can be accomplished without paging delays, at least on the Reality Engine.

The several city databases each comprise hundreds of buildings but, due to the hierarchical building structure, frame rates are unaffected until one navigates relatively close to a city. Further, only buildings in the region of the current view are paged in, and only those within the current view are rendered. Significantly higher frame rates can be attained by strategic use of LODs. The efficient fast paging and hierarchical structure ensure that even when one jumps instantaneously to an entirely new location, lower resolution data are displayed quickly and higher resolution data relatively fast.

6 CONCLUSIONS AND FUTURE WORK

We have attacked the problem of increasingly vast stores of global information for the digital earth by presenting a general approach to data organization and real-time exploration. This approach is based on a novel global hierarchical data model. Our recent work has revealed that this framework can be quite generally applied to the earth and anything on it, above it, or even below it. This includes terrain elevations, phototextures and imagery, maps, buildings, moving or flying vehicles, weather, and other data. Further, the framework provides a geospatial visual data mining approach where one can navigate continuously from global overviews to high resolution local views. The framework is also quite flexible and has been applied to a range of single and networked systems ranging from a single-processor PC to immersive systems with multiple projection screens and coupled computers.

Our approach offers a general framework for digital earth applications because:

- It accepts and integrates all types of geospatial data into a global framework.
- It is scalable to very large data stores and to distributed environments.
- It provides an interactive visualization framework.
- It supports discovery of new information via navigation in context with unfolding detail.

We also described how large collections of objects, such as buildings and weather clouds, can efficiently fit, along with terrain, into the geospatial hierarchy. Application results demonstrated that our approach is both scalable and general because it is able to handle both large scale global terrain information and multiple collections of objects (e.g., cities) placed around the earth with full interactivity and without extensive memory load. Finally, our approach shows that levels of detail can be naturally incorporated to provide improved detail management.

The geospatial data model can readily incorporate full GIS capabilities since multiple data layers are already accessed via geocodes. We are now developing a queryable GIS data structure to augment the global model. The GIS layers will be in the hierarchical format presented here to ensure fast access and display.

The approach is extendible to a networked visual data mining system. In fact we have obtained reasonable responsiveness in accessing remote global databases through the VGIS visual interface. Our flexible paging and LOD structures can be adjusted to improve response times and even tuned to the characteristics of a particular client. We are building on these capabilities by inserting an internetwork server into the VGIS architecture. Eventually we plan to develop a completely machine-independent version of VGIS by porting the system to Java3D.

ACKNOWLEDGMENTS

This work was performed in part under grants from the NSF Alexandria Project, the NSF Large Scientific and Software DataSet Visualization program, and ONR. Initial development was made under a contract from the U.S. Army Research Laboratory.

REFERENCES

Cos90 Cosman, M.A., Mathisen, A.E., and Robinson, J.A. A New Visual System to Support Advanced Requirements. *Proc. IMAGE V Conference*, pp. 370-380 (1990).

Cox97 M. Cox and D. Ellsworth. Application-Controlled Demand Paging for Out-of-Core Visualization. Proceedings, *IEEE Visualization '97*, pp. 235-244 (1997).

Dav98 Davis, D., Jiang, T.F., Ribarsky, W., Faust, N. Intent, Perception, and Out-of-Core Visualization Applied to Terrain. *Proc. IEEE Visualization '98*, pp. pp. 455-458 (1998).

Dav99 Davis, D., Jiang, T.F., Ribarsky, W., Faust, N., and Ho, S. Real-Time Visualization of Scalably Large Collections of Heterogeneous Objects. *Proc. IEEE Visualization '99*, pp. 437-440 (1999).

DeF95 De Floriani, L. and Puppo, E. Hierarchical Triangulation for Multiresolution Surface Description. *ACM Transactions on Graphics*, 14(4), pp. 363-411 (1995).

Duc97 Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M.C., Aldrich, C., and Mineev-Weinstein, M. B. ROAMing Terrain: Real-time Optimally Adapting Meshes. *Proc. IEEE Visualization '97*, pp. 81-88 (1997).

Erv93 Ervin, S.M. Landscape Visualization with Emaps. *IEEE Computer Graphics & Applications* 13(2), pp. 28-33 (1993).

Fek90 Fekete, G and Treinish, L. Sphere quadtrees: a new data structure to support the visualization of spherically distributed data. *Proc.SPIE*, 1259, pp. 242-253 (1990).

Gar95 Garland, M. and Heckbert, P.S. Fast Polygonal Approximation of Terrains and Height Fields. Tech. Rep. CMU-CS-95-181, CS Dept., Carnegie Mellon U. (1995).

Goo92 Goodchild, M.F. and Shiren, Y. A Hierarchical Spatial Data Structure for Global Geographic Information Systems. *CVGIP: Graphical Models and Image Processing* 54(1), pp. 31-44 (1992).

Gru95 Grueneis, G. Mayer, P. Sauter, J., and Schmidt, A. T_Vision. *Visual Proc. of SIGGRAPH 95*}, p. 134 (1995).

Hop98 Hoppe, H. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. *Proc. IEEE Visualization '98*, pp. 35-42 (1998).

Kim99 Kim, S.-S.; Kim, Y.-S.; Cho, M.-G. A geometric compression algorithm for massive terrain data using Delaunay Triangulation. *Proc. of 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99*, Vol. 1, pp.124-131 (1999).

Lin96 Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L.F., Faust, N., and Turner, G.A. Real-Time, Continuous Level of Detail Rendering of Height Fields. *Proc. SIGGRAPH '96, Computer Graphics*, pp. 109-118 (1996).

Lin97 Peter Lindstrom, David Koller, William Ribarsky, Larry Hodges, and Nick Faust (1997). An Integrated Global GIS and Visual Simulation System. Report GIT-GVU-97-07.

Nis93 Nishita, T., Sirai, T., Tadamura, K., and Nakamae, E. Display of the Earth Taking into Account Atmospheric Scattering. *Proc. SIGGRAPH '93, Computer Graphics*, pp. 175-182 (1993).

Res00 For more information on various environments and platforms for VGIS, see www.gvu.gatech.edu/datavis/research/research.html

Rib00 Ribarsky, W., Wasilewski, T., Jiang, T.Y., and Faust, N. Real Time Weather Data on Terrain. Submitted to *IEEE Visualization '00*, (2000).

Sam84 Samet, H. The Quadtree and Related Hierarchical Data Structures. *ACM Comp. Surveys* 16(2), pp. 187-260 (1984).

Sch94 Schroder, F. and Rossbach, P. Managing the Complexity of Digital Terrain Models. *Computers & Graphics* 18(6), pp. 775-783 (1994).

Uen97 S.K. Ueng, C. Sikorski, and K.L. Ma. Out-of-Core Streamline Visualization on Large Unstructured Meshes. *Transactions on Visualization and Computer Graphics* 3(4), pp. 370-379 (1997).

Xia96 Xia, J.C. and Varshney, A. Dynamic View-Dependent Simplification for Polygonal Models. *Proc. Visualization '96*, pp. 327-334 (1996).

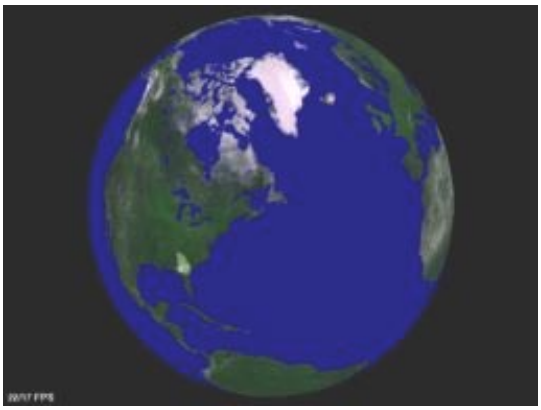


Fig. 8 World overview. This and all following scenes can be reached by continuous navigation.

Figs. 9 Views of the Grand Canyon (left) and a mountainous area in Rwanda.

Figs. 10 Views of cities (from left to right) at NTC in California, Savannah Georgia, and Atlanta. All cities are in the same database. The hierarchical object structure ensures that only those buildings in view are loaded and rendered.