International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

VIRTUAL GEOEXPLORATION: CONCEPTS AND DESIGN CHOICES

RENATO PAJAROLA*
Department of Information and Computer Science
University of California Irvine
Irvine, CA 92697

and

PETER WIDMAYER

Institute of Theoretical Computer Science
Computer Science Department, ETH Zürich
8092 Zürich, Switzerland

Received 19 September 1997
Revised 20 March 1998
Communicated by J-D. Boissonnat

ABSTRACT

A trip through a virtual world is a powerful metaphor and a natural way of interacting with a geographic information system (GIS). Virtual reality (VR) systems provide visual realism and real-time navigation and interaction for data in core, but fail to cope with very large amounts of data, and to provide the general functionality of information systems. We describe a prototype system that overcomes these problems by coupling two platforms: A client that runs the VR component interacts via a (local or wide area) network with a server that runs an object oriented database containing geographic data. For the purpose of accessing data efficiently, we describe how to integrate a geometric index into the database, and how to perform the operations that are requested in a real-time trip through the virtual world.

Keywords: interactive visualization, multiresolution triangulation, geoinformation.

1. Introduction

More than a decade ago, geographic information systems (GIS) have started to play a significant role in a large number of application domains. These applications become more and more complex and require the maintenance of an ever increasing variety and amount of data. To be useful even for a non-expert user, a GIS should allow queries to be posed in a natural way. For the geometric part of a query, it appears natural to let the user explore a virtual world of her choice — indeed quite a powerful metaphor of user interaction.[2] The user might, for instance, look around

*Email: pajarola@acm.org, pajarola@ics.uci.edu in the virtual alps to find a suitable place for her skiing vacations. This search will not only involve data on the (three-dimensional) terrain, but might also consider train connections, prices and availability of hotel rooms, among many other decision parameters. Proper visualization of the scene and all relevant data at the same time is then crucial for the ease of use of the system as a whole.[12] Such a GIS interface could in fact be one of the prime examples of a post-WIMP[a] user interface.[23]

A trip through a virtual world is useful only if it provides latency-free interaction. In particular, it must be possible to move through a terrain (for instance in a train or in an airplane) and see the changes in the scenery in real-time. This requirement for fast data delivery to the simulating VR application calls for an extraordinarily efficient GIS. In particular, the geometric terrain data must be accessed efficiently, while at the same time they should be maintained in the GIS in order to support typical database functionalities, such as transactions and recovery. We propose a way to achieve this efficiency.

The system we propose, ViRGIS (for *Virtual Reality GIS*), maintains three-dimensional terrain data in vector form (such as surface triangulations), raster data (such as those from satellite images and topographic maps), and non-geometric data (such as population counts of cities). It allows a user to move through the scene in real-time by means of a standard input device such as a mouse, and to interact with the GIS (through a point-and-click interface with pop-up windows for non-geometric data). Thus far, we have not implemented virtual reality in- and outputs in the strong sense, such as interaction via 3D input devices or head-mounted displays; our interface is a desktop VR or video user interface.[1] The basic architecture and concept of ViRGIS also served as a framework for other VR information systems, e.g. in tourism.[22]

A number of software tools are available that support real-time walk-through in three-dimensional scenes. Usually, current software tools for this purpose limit their walk-through capabilities to a scene that is loaded once and stored in main memory. This is also true for the *IRIS Performer Toolkit*[b], which we used in our prototype. The scene as a whole (but not parts of it) can be replaced by another one. While this is good enough for several applications, it is far from satisfactory for geo-exploration, where gigabytes (or even terabytes) of data need to be explored. That is, while the walk-through progresses, data must be loaded dynamically from disk. Our virtual reality component implements such a dynamic maintenance of a part of a very large scene in main memory, to be visualized using the IRIS Performer Toolkit. To reduce the graphics load, ViRGIS displays exact data only where they are useful. In particular, a user should see all the available details of a scene when she can see them in reality, that is, in the close neighborhood of the viewpoint. The farther away we look, the fewer details we can see in reality and need to see in a virtual world; the resolution of the retina and the resolution of the screen both coherently provide an *image space* that limits the amount of detail. We therefore propose to maintain data in various *levels of detail* (LOD) and access only those

[a]WIMP stands for windows, icons, menus, and a pointing device

[b]The *IRIS Performer Toolkit* is a product of *SiliconGraphics, Inc.*

levels that are needed. Lower resolution and fewer details are satisfactory not only far from the viewpoint, but also when a flight over a scene is so quick that the eye cannot capture details anyway; this physiological feature corresponds nicely to a reduced graphics load when the image needs to change rapidly. By using the LOD concept to our advantage, we arrive at a sufficiently low amount of graphics data per time unit to be displayed, for typical graphics engine capacities (for instance, an SGI Indigo2 IMPACT 10000 can display up to 652k textured and Gouraud shaded triangles per second).

To efficiently retrieve the data in the database and bring it into the visualization component, we propose storage and retrieval schemes for location-oriented and LOD-driven dynamic loading. All geometric objects except the terrain (its geometry, and its textures) are maintained in a hierarchical geometric index, for instance an *R-tree*.[11] It turns out to be appropriate to incorporate the geometric index into an object oriented database system (*ObjectStore*[c]), thus combining efficient access with all the advantages of object orientation. The geometry of the terrain is stored in a restricted quadtree, taking LODs into account; the textures are maintained in simple hashing structures.

Since the GIS data might reside on a workstation to which many users' workstations are connected, we propose to run the database on the database server, while the graphics software (such as the IRIS Performer Toolkit) runs on the user side, for more details see Refs. [18]. Both parts of ViRGIS might of course run on the same machine, but will in general cooperate over a network; in particular, we are currently implementing a WWW interface to a GIS server at ETH Zurich with terrain data of Switzerland[d].

The remainder of this paper is organized as follows. The next section describes the overall system architecture. Section 3 discusses the integration of a spatial access structure, together with the physical clustering, with ObjectStore. Section 4 briefly explains the level-of-detail concept. Section 5 proposes a way of using multiresolution terrain modeling for realizing levels of detail, and Section 6 describes a data structure for that purpose. Section 7 concludes the paper.

2. Architecture of ViRGIS

In current VR applications, the interaction with large worlds, including their visualization, is still an efficiency problem whenever the memory requirements exceed available main memory (and even exceed swap space). This is one of the reasons for combining in ViRGIS two loosely coupled components, one for visualization and one for data handling. Figure 1 shows the overall system architecture of ViRGIS in which these two components communicate over a (local area or wide area) network. Multiple clients are allowed to connect to a database and explore the same data set. Each client can independently request data corresponding to its needs, especially to adjust the visual part of its local scene to the user's movements. Different

[c] *ObjectStore* is a registered trademark of *Object Design, Inc.*
[d] Further information on this upcoming WWW-interface can be obtained from virgis@inf.ethz.ch.

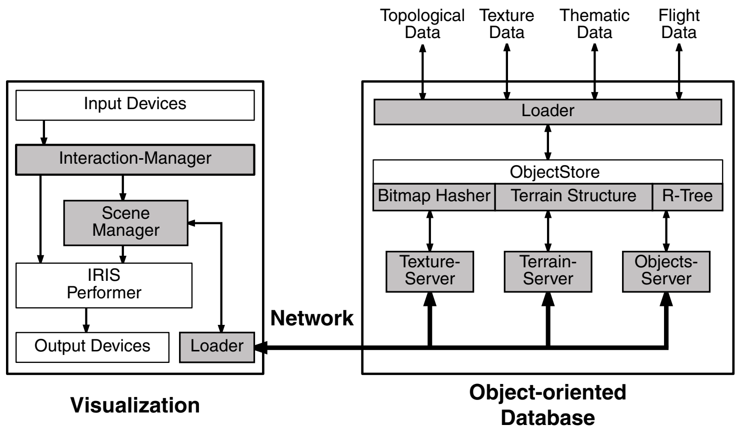

Fig. 1. System architecture

databases could serve queries for different regions, for example one database-server per country. The main data types which are incorporated in the system are the digital elevation model and the texture data, which can be satellite images, aerial photographs or cartographic pixel maps. Other data types are flight-trajectories and thematic information connected to a point location on the terrain; more types can be added as needed.

The database component maintains the texture images, terrain data, and other geometric objects in different spatial access structures within a database; it will be described in Section 3. All data structures, for geometries, the terrain and texture data, support arbitrary rectangular two- or three-dimensional range queries. A server application provides network access to each database. It opens the required database for the different data types and then listens to a socket port on the network for incoming requests. Each request is transformed into a query on the data structures, and the result is written back on the network in a predefined format.

In the visualization component, most of the work is performed by the *scene manager* which takes care of updating the actual scene dynamically according to the user's movements, as shown in Figure 2. The actual scene is only the currently visible part of a much larger virtual world, stored in the database, and it is subdivided into a set (i.e. a matrix) of rectangular patches. New rows or columns of patches are loaded from the database on demand; outdated or invisible ones are discarded. The scene manager also takes care of the correct LOD for every patch, and keeps track of which one is already loaded and which one has to be requested or updated from the database due to user movements or changed parameters from the user-interface. Such a regular subdivision of the visible space eases the interaction between visualization and data management; it also efficiently supports culling of invisible patches. Each patch has different data associated with it: texture images, terrain data and other geometric objects. The texture images are maintained as such, but the terrain data is stored in a restricted quadtree for each patch. This multiresolution data structure is explained in more detail in Section 6.

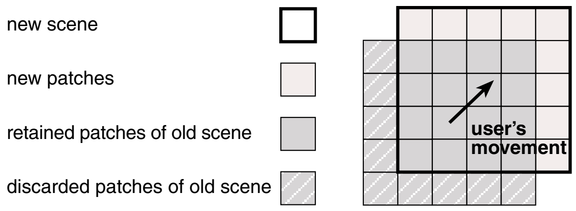

Fig. 2. Dynamic scene update

3. Spatial Access Within an OODBMS

Spatial access structures must support the dictionary operations (insertion, deletion, exact match query) as well as *proximity queries* based on the geometric neighborhood of objects. An important type of proximity queries are *range queries*: They ask for all objects whose geometric key intersects a given query range. In our application, rectangular range queries are used by the scene manager to read new parts of the world scene into the actual scene. To support proximity queries efficiently, a spatial access structure clusters objects that are close in space into blocks on external storage. Spatial access structures and their applications are discussed in many places in the literature, see Refs. [8,16,19,20].

We have chosen an R-tree[11] to store flight trajectories and other scattered objects with geometric keys. It provides not only good range query performance, but it also prevails by its simplicity and conceptual clarity. The R-tree can be seen as a modified B^+-tree with *rectangular regions* as keys. Each node of the tree represents a subspace of the universe, with the root representing the entire universe. An inner node covers its subspace with rectangular regions; they are represented by the nodes of the next level in the tree. Each leaf node represents a block of objects. For each node, the geometric key is the bounding box of the geometric keys of all objects in the subtree rooted at this node (for inner nodes) or of all objects in its block (for leaves).

Implemented as a file, each R-tree directory node as well as each data block is maintained as one external storage block. In this way, the contents of a node or a data block can be read or written with one block access. Now, when we embed the R-tree (or some other hierarchical access structure) into a database system efficiently, we request the following:

(i) The database system must preserve the physical clustering of objects or entries in a data structure, i.e. objects or entries grouped in a node of the data structure must also be grouped in a block in the database system.
(ii) The database system must allow access to a single block, avoiding unnecessary overhead in time, as caused by reading several unneeded blocks at once.

That is, for an efficient integration of an access structure with a database system, the latter must allow the user to directly control the clustering and access patterns used for storing physical blocks. While relational DBMSs tend to limit the user in this respect, many object-oriented database management systems (like *ObjectStore*) comply. *ObjectStore* is a commercially available object-oriented database system with the usual DBMS features such as transactions, recovery, and concurrency control.

Each *ObjectStore* database is divided into *segments*, and these again may have *object-clusters* as subdivisions. In contrast to object-clusters, whose size is fixed, segments are flexible in their size, and grow or shrink according to the data stored in them. Figure 3 shows the physical memory layout in terms of the usage of segments and object-clusters for the implemented R-tree. Each complete R-tree structure is stored in one segment, and each of its nodes covers exactly one object-cluster within that segment. The R-tree object *RTreeRoot* is not tied to any object-cluster, because it doesn't hold any data, but merely interfaces to applications.

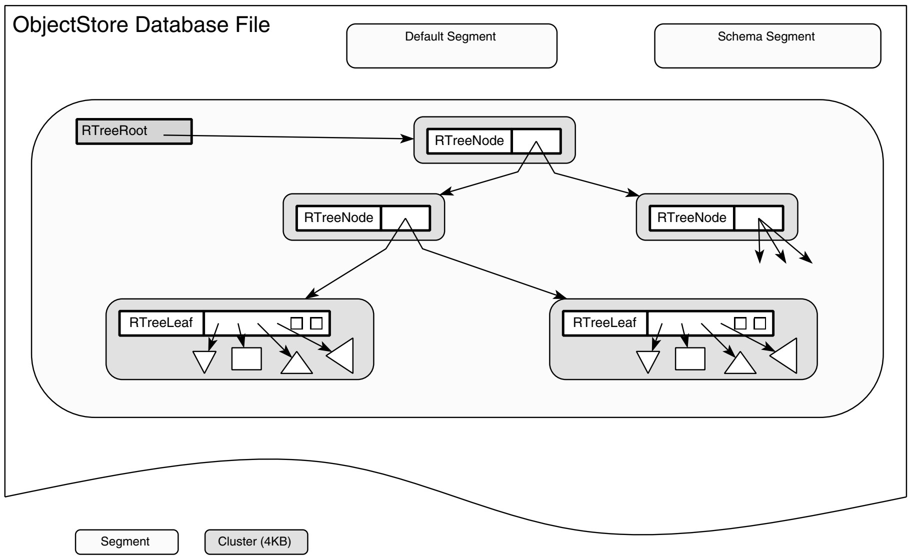

Fig. 3. Clustering

For maintaining the terrain data of the digital elevation model, essentially a matrix of height values (also a height field), we use a quadtree[19,20] data structure. We are also experimenting with other access structures; hashing structures in particular might provide fast access to such regularly gridded data. Superimposed on the height field, there is a dependency-graph of the restricted quadtree triangulation discussed in Section 6. This permits combining range queries with a level of detail or error driven selection of elevation points. The result of a query can be returned in the form of a restricted quadtree which is used by the visualization component to represent and display the surface of one terrain patch.

Raster and image data is stored in a two-dimensional hashing structure which also handles overlapping textures at different resolutions. Each image (or raster data-set) is partitioned into a set of rectangular patches representing the smallest access units to this image. A hash-directory – a matrix with pointers – allows for fast look-up of every image patch.

4. Taking Advantage of Levels of Detail

The appropriate use of different levels of detail (LOD) for different parts of the visible scene can significantly reduce the number of geometric primitives which have to be rendered. Simply put, a larger scene with fewer details can be displayed at the same frame-rate as a smaller one with more details. Furthermore, the use of different LODs can also lead to an enhanced realistic impression of the scene, since the farther the objects are, the lower we choose their LOD. The LOD strategy can also differ between applications: For instance, the *view-point centered* strategy gives lower LODs to objects which are farther away from the view point, whereas the *view-direction oriented* strategy assigns lower LODs to objects that are farther away from the ray of the view direction.

In ViRGIS, the visualization uses a combination of view-point centered and view-direction oriented LOD strategy. Figure 4 shows an example of a resulting LOD surface, or error distribution, where the viewing direction points from the center to the lower right. The center below the view point has the lowest error, or the highest LOD, of the displayed terrain approximation. The high LOD, or low error-rate, runs along the view direction, and the accuracy gradually decreases with the distance to the view point and view direction.

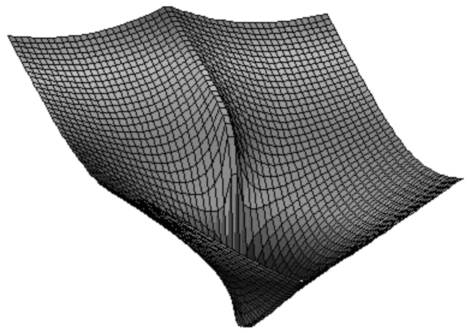

Fig. 4. Error distribution

For efficiency reasons we associate one LOD value with each patch of the scene as shown in Figure 5, even though our application provides continuous LODs. This value can change from frame to frame, as the user moves through the scenery. For every frame, the scene manager recalculates the LOD values for every displayed patch. Whenever the approximation-error difference between the actually loaded or selected data and the calculated value exceeds a certain threshold, the respective patch is scheduled for an update. Updating a patch means that the geometry is reconstructed for the new LOD value. This can be done in two ways. If the patch already provides the data necessary for the new LOD, then the new triangulation is generated from the quadtree representing the patch inside of the visualization component. If, however, some points are missing, first a range-query with the appropriate error-range is sent to the terrain database, and the result is incorporated into the patch's quadtree. Then, the correct triangulation can be constructed. For this to work, the database must provide an *error-range* query such that the visualization can incrementally update every single patch.

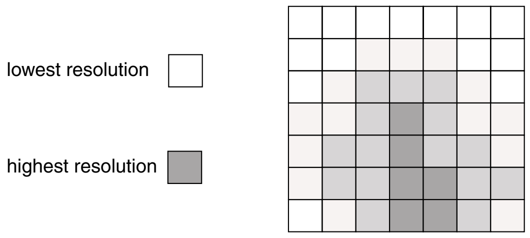

Fig. 5. LOD strategy

5. Multiresolution Terrain Models

There is a range of problems which arise when considering LODs for a digital terrain model. First of all, the different LODs could be represented as uniform height fields, as a discrete number of separate triangulations, or as dynamic point sets or triangulations which are updated according to the LOD parameter. Depending on the representation, the relevant terrain points for a specific LOD must be selected. In the first case, sub-, super-sampling and interpolation can be used. The second case allows any type of selection, if a suitable triangulation can be computed on it. In the last case, the selection has to be related with the triangulation. Height fields are easy to maintain, both on the database side and for visualization. However, they do not adapt to the actual terrain, and therefore to the visual impression. That is because the selection of points just depends on their position and not on the terrain model itself; smoothing and omission of relevant details are the consequences. Maintaining a fixed set of different triangulations for case two allows fast selection of the triangles for one specific LOD out of a fixed number of predefined LODs. Many different algorithms of how to convert a regular grid of height points into a surface triangulation, also called a *triangulated irregular network* (TIN), for a specific LOD can be found in the literature, see Refs. [10,13,14]. However, no continuous LODs can be supported in this way, and also the data storage costs are very high. Providing a terrain representation which supports continuous LOD selection and produces a terrain-adaptive triangulation is quite hard, because to do so, the triangulation has to be constructed on demand for every selection (this can be very time-consuming). This is also true for hierarchical triangulation models,[6] which also have problems to efficiently answer rectangular range queries.

Because the desired LOD changes as we get closer to a location, it should be efficient to dynamically load additional data from the database in order to refine a region that is already in core at a low resolution. This goal of incremental loading is difficult to achieve for surface triangulations,[3,4,5] because different LODs in adjacent triangles may lead to cracks in the terrain, unless special care is taken to neatly stitch adjacent triangles together. Figure 6 shows how cracks can easily be avoided when using a regularly gridded height field. However, this is far more complex when using TINs.

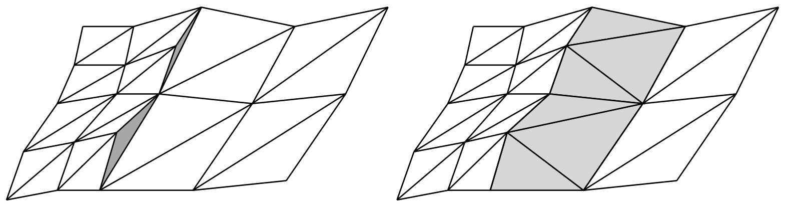
Fig. 6. Cracks between different LODs

Hierarchical multiresolution triangulations provide some compromises between a good triangulation with well shaped triangles, and continuous LOD transitions to smoothly adapt to the terrain. To guarantee a matching triangulation where all edges pertain to the boundary of exactly two triangles, a triangle cannot be subdivided independently of its neighbors. A subdivision due to some LOD criteria can lead to resolving subdivisions in neighboring triangles. In Figure 7, the initial subdivisions of triangles A and B to achieve a certain approximation would not match. Therefore, the subdivision of triangle B is changed to B' to match. In general, mismatches are not restricted to appear among siblings in the hierarchy, or in a local neighborhood: The mismatch between triangles C and D in Figure 7 is an example. This matching problem gets even worse when independently triangulated terrain patches are placed side by side, because retriangulation of border triangles of neighboring patches would be necessary.

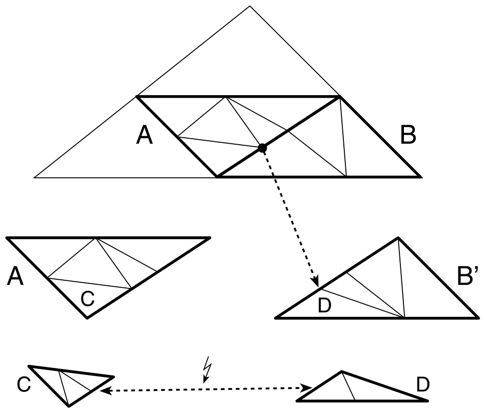

Fig. 7. Hierarchical triangulation

The different multiresolution models can be useful in several application domains. We have recently developed a World Wide Web interface to our terrain database using the raster digital elevation model. In this application, not the approximation quality is essential, but the response time to user interactions, and the drawing speed. However, this grid based approach is not satisfactory for more sophisticated visualization systems because of its lack of terrain adaptivity. Even using a fixed set of different LODs, TINs of different approximation quality turned out to be slow in network transmission, storage consuming, and difficult to achieve a matching multiresolution triangulation. Therefore, in the current version of ViRGIS we use the *restricted quadtree triangulation*[21,24] as the underlying multiresolution model; it will be described in more detail in the next section. This approach does not suffer from any of the problems mentioned so far. It has low storage costs because the $x-$ and $y-$coordinates are given implicitly, and no explicit triangulation topology has to be stored in the database. This also leads to a fast network transmission where the $x-$ and $y-$coordinates can be omitted too, and where the triangulation is implicitly given by the order of the points. Each patch is represented as one restricted quadtree with a terrain-adaptive triangulation of a particular LOD. This adaptive triangulation is, by definition, matching within the patch boundaries. Furthermore, continuous and smooth LOD transitions between all patches of the visible scene can be achieved with an additional reconciliation stage. For further technical details and complexity analysis refer to our technical report in Ref. [17].

6. Restricted Quadtree

The basic idea of the restricted quadtree was first used for parametric surfaces[24] and later for terrains.[15] We can directly apply the restricted quadtree method to the terrain data itself, and not to an intermediate parameter space. Additionally, in comparison to the first application to parametric surfaces[24] we need fewer propagated subdivisions. We also have a different LOD measure than the other terrain rendering system[15] that uses the restricted quadtree too. Their LOD measure is based on screen-resolution in the image space; we base our measure on the approximation of the best model, using all points in the height field. Furthermore, we just retriangulate a patch whenever its change in LOD exceeds a certain threshold, and not for every frame. In our dynamic scene management, we additionally have to avoid cracks between different patches. This is done by mutually exchanging the missing points on the edge between two adjacent patches.

The subdivision of a basic block in a restricted quadtree has two stages, as shown in Figure 8. The two initial triangles are first split into four by adding the center point c, which lies on the next refinement of the grid. In the next stage the middle points of the sides of the initial square can be added, forming four new basic blocks. These points do not have to be added all at once. As soon as two neighboring points such as a and b in Figure 8 are added, subdivision can proceed for block A. To guarantee a matching triangulation, cracks have to be avoided. When adding point p in Figure 8, the center point of the adjacent block has to be added too, regardless of the approximation error subdivision criterion of this block. Indeed, every point depends on two others to guarantee a matching triangulation; this is shown in Figure 9. Whenever a point is selected to be included in the triangulation, all its dependents have to be included too. The propagation stops when no more dependents exist, or when they are already selected.

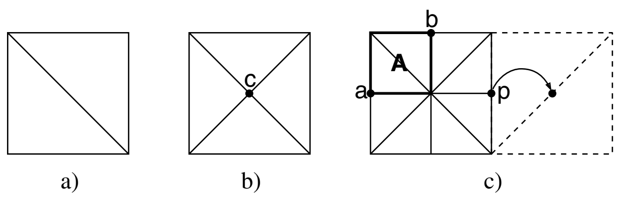

Fig. 8. Quadtree subdivision

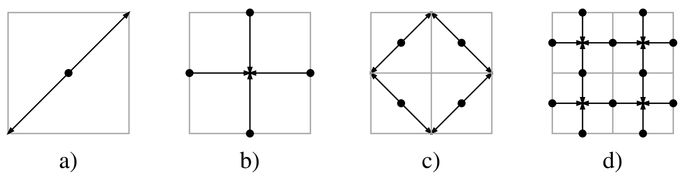

Fig. 9. Dependency graph

The triangulation of a restricted quadtree is very efficient and is given implicitly; no geometric computations have to be performed. All triangles are isosceles and have a 90° angle. The quadtree can be triangulated recursively, where only cases as in Figure 8 a) and b) can occur at different scales and orientations. The triangulation of a restricted quadtree as shown in Figure 10 a) can be represented as one single linear sequence of triangles, a so-called triangle strip; triangle strips are efficiently supported by computer graphics software libraries and hardware. Figure 10 b) shows a triangle strip that can be computed from a restricted quadtree. This triangle strip can be constructed in linear time by a recursive traversal of the restricted quadtree.[17]

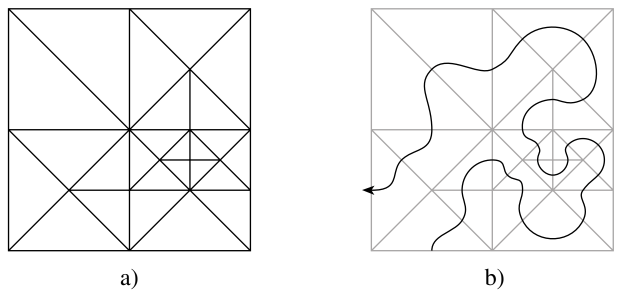

Fig. 10. Restricted quadtree triangulation and triangle-strip generation

As mentioned in Section 2, the visible scene is divided into patches, each of which is represented as one restricted quadtree in the visualization component. Such a patch at a certain LOD range is also the unit of access to the terrain database. The visualization component does not have to construct a restricted quadtree; each patch is readily transmitted as such. Efficient network transmission is achieved by sending the quadtree elements in a depth-first tree traversal order. Note that transmitting a quadtree is much more efficient than communicating a TIN, because no $x-$ or $y-$coordinates are needed: they are implicitly given through the element's position in the quadtree. Furthermore, the triangulation topology is also implicitly given by the restricted quadtree triangulation rules and must not be conveyed between the database and visualization components.

Therefore, the database must support extraction of a restricted quadtree according to a two-dimensional query range, the area of the required patch, and based on an approximation error, or LOD range. Points are consistently selected from the data structure holding the elevation data to form a restricted quadtree. Thereby, the database and the visualization component use the same restricted quadtree dependency graph. Many spatial access structures for points, together with the superimposed dependency graph will provide efficient point selection. However, the efficiency in access time and storage costs might differ quite a bit. See also our technical report in Ref. [17] for a proposal of a very efficient storage and retrieval structure to maintain a height field and support extraction of restricted quadtrees.

7. Conclusion

We have proposed an efficient visualization system for large scale terrain data. It turned out that we cannot simply glue together known (optimal) solutions for the various algorithmic problems that an integrated system must solve. Knowledge from different fields, such as computer graphics, computational geometry, database systems and spatial data structures, needs to be interwoven intricately to arrive at a practical overall system. The integration proposed by ViRGIS implies that the database as well as the visualization component can be realized by any suitable GIS or visualization toolkit. Current advanced GIS such as *Paradise*[7] and *Sequoia 2000*,[9] focusing on efficient storage and retrieval, are ideal candidates for the data management part of the ViRGIS architecture.

The virtual reality metaphor of user interaction embodied in ViRGIS will have to prove its fitness in different application scenarios, such as flight simulation and management of flight trajectories, land planning and ground management, environmental compatibility tests and interactive tourism information systems.

Acknowledgments

ViRGIS is a joint project of groups at the MultiMedia Laboratory at the University of Zürich and the Institute of Theoretical Computer Science at ETH Zürich. The current version of the software was written with the help of the diploma students P. Aschwanden, F. Kägi, S. Odendahl, M. Roth, S. Verschären and L. Vonblon. We gratefully acknowledge the support of the Swiss National Science Foundation and the European Community.

References

1. P. W. Agnew and A. S. Kellerman. *Distributed Multimedia – Technologies, Applications, and Opportunities in the Digital Information Industry: A Guide for Users and Providers.* ACM Press and Addison Wesley, Reading, Massachusetts, 1996.
2. R. Coyne. *Designing Information Technology in the Postmodern Age.* The MIT Press, Cambridge, Massachusetts, 1995.

3. M. de Berg. Visualization of TINs. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems, Summerschool, Udine*, volume 1340 of *Lecture Notes in Computer Science*, pages 79–97. Springer-Verlag, 1997.

4. M. de Berg and K. Dobrindt. On levels of detail in terrains. In *11th Symposium on Computational Geometry*, pages C26–C27. ACM, 1995.

5. L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. *The Visual Computer*, 12(7):317–345, August 1996.

6. L. De Floriani and E. Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics*, 14(4):363–411, 1995.

7. D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu. Client-server paradise. In *Proceedings of the 20th VLDB Conf.*, pages 558–569, 1994.

8. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 1997. to appear.

9. K. Gardels. Sequoia 2000 and geographic information: The guernewool geoprocessor. In T. C. Waugh and R. G. Healey, editors, *Proc. 6th Int. Symposium on Spatial Data Handling*, volume 2 of *Advances in GIS Research*, pages 1072–1085. Taylor & Francis, London, 1994.

10. M. Garland and P. S. Heckbert. Fast polygonal approximation of terrains and heigt fields. Technical Report cmu-cs-95-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1995.

11. A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proceedings of the* 13^{th} *ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM, 1984.

12. H. M. Hearnshaw and D. J. Unwin, editors. *Visualization in Geographical Information Systems*. John Wiley & Sons, Chichester, 1994.

13. M. Heller. Triangulation algorithms for adaptive terrain modeling. In *Proc. 4th Int. Symposium on Spatial Data Handling*, volume 1, pages 163–174, 1990.

14. J. Lee. Comparison of existing methods for building triangular irregular network models of terrain from grid digital elevation models. *International Journal of Geographic Information Systems*, 5(3):267–285, 1991.

15. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings SIGGRAPH 96*, pages 109–118. ACM SIGGRAPH, 1996.

16. J. Nievergelt and P. Widmayer. Spatial data structures: Concepts and design choices. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems, Summerschool, Udine*, volume 1340 of *Lecture Notes in Computer Science*, pages 153–197. Springer-Verlag, 1997.

17. R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. Technical Report 292, Dept. of Computer Science, ETH Zürich, 1998. ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/292.ps.

18. R. Pajarola, T. Ohler, P. Stucki, K. Szabo, and P. Widmayer. The alps at your fingertips: Virtual reality and geoinformation systems. In *Proceedings 14th International Conference on Data Engineering, ICDE '98*, pages 550–557. IEEE, 1998.

19. H. Samet. *Applications of Spatial Data Structures: computer graphics, image processing, and GIS*. Addison Wesley, Reading, Massachusetts, 1989.

20. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison Wesley, Reading, Massachusetts, 1989.

21. R. Sivan and H. Samet. Algorithms for constructing quadtree surface maps. In *Proc. 5th Int. Symposium on Spatial Data Handling*, pages 361–370, August 1992.

22. K. Szabo, P. Stucki, P. Aschwanden, T. Ohler, R. Pajarola, and P. Widmayer. A virtual reality based system environment for intuitive walk-throughs and exploration of large-scale tourist information. In *Proceedings Enter95: Information and Communication Technologies in Tourism*, pages 10–15. Springer-Verlag, Vienna, 1995.

23. A. van Dam. Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67, February 1997.

24. B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proceedings SIGGRAPH 87*, number 4 in ACM Journal Computer Graphics, pages 103–110. ACM SIGGRAPH, 1987.