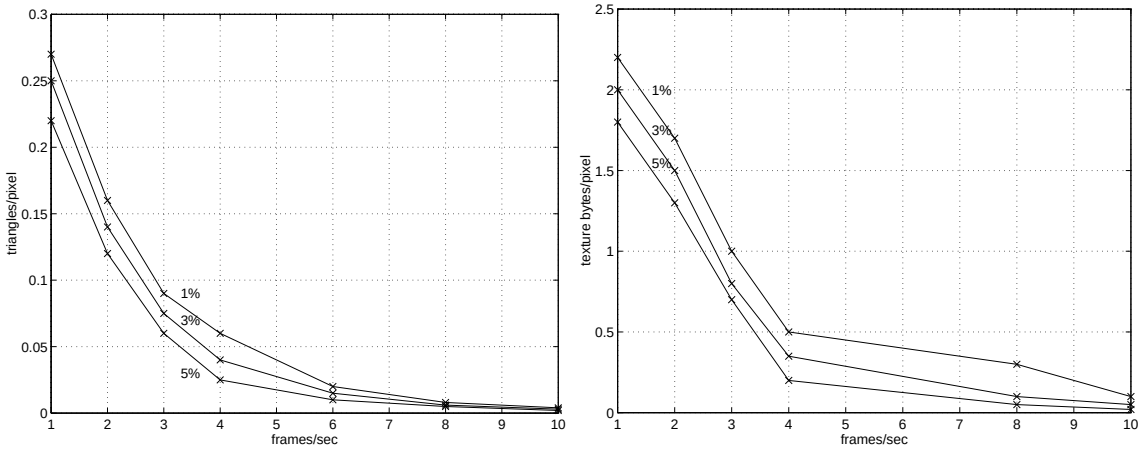


(a) (b)

Figure 5: Speed/resolution tradeoff in our prototype visualization client while rendering 300x400 pixel images on a R5000 SGI O_2, accessing the scene database server over a 3 KByte/sec network. (a) Varying only geometric resolution. The texture resolution is fixed to 0.5 compressed texture bytes per pixel. (b) Varying only texture resolution. The geometric resolution is fixed to 0.06 triangles/pixel. The individual curves correspond to different flight velocities, which influence the turnover of data in system caches and bandwidth overhead.

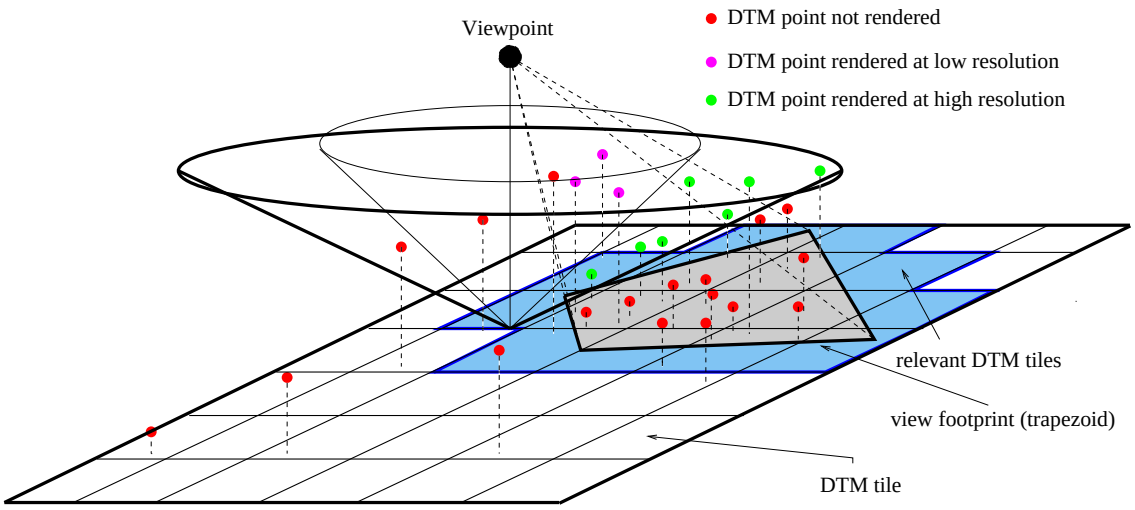


Figure 3: Determining the DTM points of the rendered Delaunay triangulation for a given view at different geometric resolutions. The narrow cone represents a low-resolution view, and the wide one a high resolution. The "elevations" of the DTM points are their precalculated grades. All points within the footprint with grade above the relevant cone are included in the triangulation. This range-reporting operation is performed efficiently using an octree structure on the points in each tile. Note that more points are admitted in the view foreground than in its background.

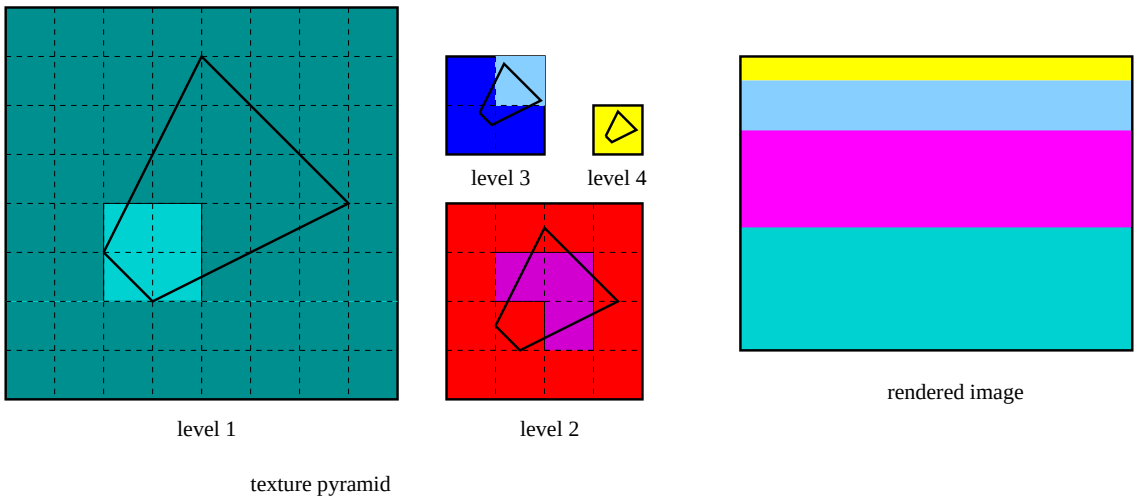
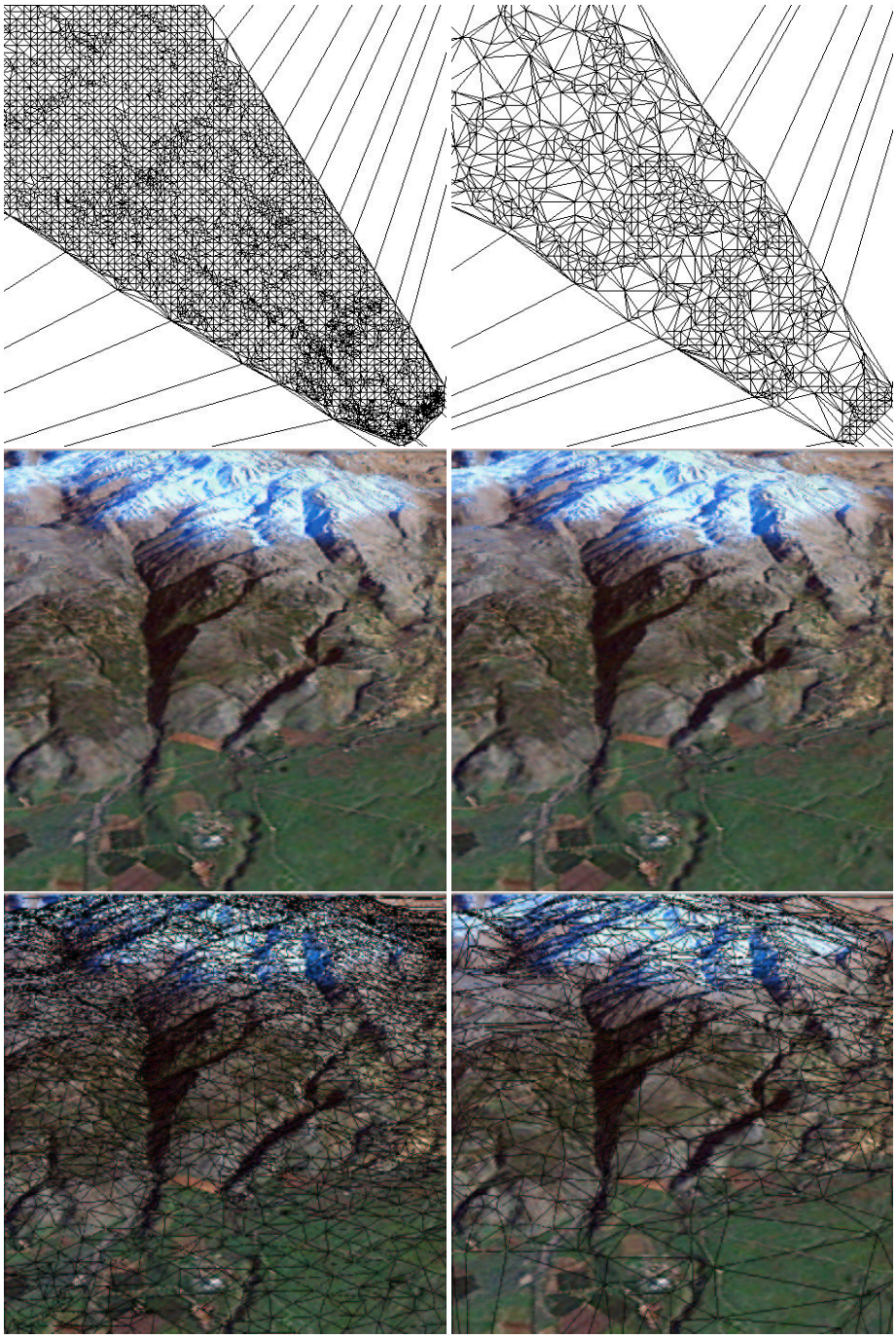


Figure 4: The contribution of individual tiles in the texture buffer to the rendered image corresponding to the marked footprint. Those tiles not contributing need not reside in the texture buffer at all, and are not streamed and decoded from the server.



(a) (b)

Figure 2: Terrain meshes (Delaunay triangulated) and views rendered at different data resolutions. (a) High resolution: 0.08 triangles/pixel and 1 texels/pixel. (b) Equivalent quality at lower resolution: 0.02 triangles and 0.8 texels/pixels. Note how more DTM points are used in foreground areas or areas of high curvature.

References

[1] M. De Berg and K. Dobrindt. On levels of detail in terrains. In *11th Annual ACM Symposium on Computational Geometry*. ACM, 1994.

[2] R.W. Buccigrossi and E.P. Simoncelli. Progressive wavelet image coding based on a conditional probability model. In *Proceedings of Int'l Conf. Acoustics Speech and Signal Processing*. IEEE, 1997.

[3] D. Cohen and C. Gotsman. Photorealistic terrain imaging and flight simulation. *IEEE Computer Graphics and Applications*, 14(2):10–12, March 1994.

[4] D. Cohen-Or, U. Lerner, E. Rich, and V. Shenkar. A real-time photo-realistic visual flythrough. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):255–265, September 1996.

[5] D. Cohen-Or and Y. Levanoni. Temporal continuity of levels of detail in Delaunay triangulated terrain. In *Proceedings of Visualization '96*. IEEE Computer Society Press, 1996.

[6] T. Delepine. Online terrain level-of-detail. In *Proceedings of ITECH*, 1997.

[7] O. Devillers, S. Meiser, and M.Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2:55–80, 1992.

[8] L. De Floriani. A pyramidal data structure for triangle-based surface representation. *IEEE Computer Graphics and Applications*, 9(2):67–78, 1989.

[9] P. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, School of Computer Science,Carnegie Mellon University,Pittsburg ,PA , 15213, 1995.

[10] K. Kaneda, F. Kato, E. Nakamae, T. Nishita, Tanaka, and Nogushi. Three-dimensional terrain modeling and display for environmental assessment. *Computer Graphics (Proceedings of SIGGRAPH'89)*, 23(3):207–214, 1989.

[11] R. Klein and T. Huttner. Simple camera-dependent approximation of terrain surfaces for fast visualization and animation. In *Proceedings of Visualization '96 (late breaking topics)*. IEEE Computer Society Press, 1996.

[12] P. Lindstrom, D. Koller, L.F. Hodges W. Ribarsky, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of SIGGRAPH '96*, 1996.

[13] M. Shamos and F. Preparata. *Computational Geometry*. Springer, 1989.

[14] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–258, 1996 (Proceedings of Eurographics, Poitiers, France, August 1996).

1. Calculate view frustum and bound terrain footprint by rectangle.
2. Scan-convert the rectangle and for each geometry tile in it:
   (a) If the tile is not in the footprint, but was in it in the previous frame, then:
      - Remove all its points from the Delaunay triangulation.

   (b) If the tile is in the footprint, but was not in the previous frame, then:
      - Request tile from server at appropriate resolution.
      - Search in tile octree for appropriate voxels.
      - Insert the points from these voxels in Delaunay triangulation.

   (c) If tile is in the footprint and was also in the previous frame, then:
      - Search in tile octree for appropriate voxels.
      - Find difference from previous frame.
      - Insert (Delete) difference points in (from) Delaunay triangulation.
3. For each texture tile in the bounding rectangle:
   (a) If the texture tile is in the footprint, but was not in the previous frame, then:
      - Calculate required resolution.
      - Request the appropriate bit stream prefix from the server.

   (b) If texture tile is in the footprint, and was also in the previous frame, then:
      - Calculate its resolution.
      - If this resolution is higher than that of the previous frame, then request more of the bit stream from the server.
4. For every tenth frame check the actual performance (frames/sec) against the required performance and adjust the geometric and/or texture resolution parameters to achieve that performance.
5. Render image.

Figure 1: Pseudo-code of the client algorithm.

olution (the lower the required resolution, the less bits required). In any case, we use any bits available at rendering time, even though there might be less than required (if the network temporarily slows down). Which tiles are relevant can be easily determined from the geometry of the footprint. Occasionally, it is neccesary to shift the contents of the texture buffer, due to the movement of the viewpoint.

5 Experimental Results

We have implemented the procedures described in Sections 2 - 4 as a prototype client/server system, the client running on a R5000 SGI O_2, at 180MHz with 64MB RAM, based on the OpenGL API, and an X/Motif GUI. This client accesses the scene database server over a 3 KByte/sec network. The main parameters influencing the overall performance of the system are the size of the visualization window, i.e. the number of rendered image pixels, and the flight velocity. This performance is measured in the client frame rate, and the quality of the imagery delivered at that frame rate. There is an obvious tradeoff between the two, which is controlled by two independent "resolution" parameters, one for geometry, and one for texture. Increasing these parameters increases the number of triangles and/or texture bytes used for the rendering process, thus increasing the image quality, but decreasing the frame rate, due to higher rendering and bandwidth overhead. There is, however, a point beyond which the resolution parameter saturates, i.e. the marginal increase in image quality is insignificant.

The geometric resolution parameter, namely, the average number of triangles rendered per image pixel, is controlled by the angle of the cone used for culling DTM points, as described in Section 3.3. The smaller the angle, the narrower the cone, admitting less DTM points into the Delaunay triangulation, in turn implying less triangles for the same number of image pixels (see also Fig. 3). The texture resolution is controlled by specifying the fraction of the texture tile bit stream imported and decoded to texels for the foreground image pixels. The resolution of the background image pixels is derived from this.

Keeping the resolution parameters and velocity fixed causes the system to maintain a fixed frame rate. Increasing the velocity would slow down the system, as the turnover of points in the Delaunay triangulation and turnover of texture tiles in the texture buffer increases, incurring more CPU and bandwidth overhead. By trial and error, it seems that reasonable image quality is obtained at a geometric resolution of 0.06 triangles and 0.5 texture bytes per output image pixel. Any more than that imposes an unneccesary load on the system, slowing it down, and any less than that results in poor quality images (see Fig. 2). A telltale sign of insufficient geometric resolution (triangles per image pixel) is if there are "jumps" (also known as "popping") in the terrain surface during animation, due to the triangles being too large and crude. A telltale sign of insufficient texture resolution (texels per image pixel) are blurred images.

Fig. 5 shows the speed/quality tradeoffs we are able to achieve with our system at different "flight" velocity parameters, when one of the geometric/texture resolution parameters is fixed, and the other varied. Velocity is measured as the percentage of non-overlapping area between footprints corresponding to successive frames. The figure shows that approximately 3 frames/sec are achievable with reasonable quality, when the image size is fixed at 300x400 pixels, and flying at an average (3%) velocity. Higher velocities result in a larger turnover of geometry and texture, slowing down the system frame rate. Our system accesses a scene database server covering the northern part of Israel, containing a total of 10^7 DTM points and 10^8 texels. The client uses a geometry cache of size 2MB RAM, and texture buffer of 1024x1024 texels.

6 Conclusion

In the long-term, our techniques will support client/server terrain visualization applications over the Internet. A large scene database resides at a central server site, and is accessed (perhaps simultaneously) by a number of low-end clients over the Internet for visualization purposes. This application requires tight optimization of the available network bandwidth and client rendering power.

The ever-increasing user appetite for larger and richer geometric scenes has forced computer graphics practitioners to develop output-sensitive rendering algorithms whose computational complexity is not sensitive to the complexity of the input scene, rather to the complexity of the output image. We have implemented this for the terrain visualization application by rendering at geometric and texture level-of-detail which changes continuously along the spatial and temporal dimensions. Our algorithm satisfies almost all of the five requirements from such an algorithm, as formulated in [12].

Use of other sophisticated data optimization techniques, such as *occlusion culling* [14], in which large portions of the geometry inside the view frustrum are efficiently culled because they are invisible, can further reduce the rendering load.

Temporal aliasing sometimes occurs in our implementation. The use of *morphing* techniques to alleviate this, such as that of Cohen-Or and Levanoni [5], are not directly applicable, again due to the dynamic nature of our Delaunay triangulation. Alternatives are being investigated.

Acknowledgements

We thank Olivier DeVillers for providing code implementing the algorithm of [7], Paul Heckbert for code implementing the algorithm of [9], and R. Buccigrossi for code implementing the algorithm of [2].

This research was supported by the Technion V.P.R. Fund - Promotion of Sponsored Research.

record of fixed length. This hierarchical spatial data structure will enable efficient range reporting of points.

3.2 View Frustum Culling

The first step in frame generation is to determine which DTM tiles are relevant to the current view. In principle, if the terrain surface were planar, the intersection of the viewing frustum with the terrain surface (the view *footprint*) would be a trapezoid, whose four vertex positions could be easily computed (see Fig. 3). Since the terrain surface is not planar, the footprint terrain is bounded by a region which is the union of *two* trapezoids, formed on horizontal planes whose elevations coincide with the minimal and maximal elevations in the projection area, repectively.

The footprint is "scan-converted" by the client to determine which DTM tiles intersect it, and what resolution data (which levels of the octree) are required. This data is requested from the server. For every tile received, the octree structure of its points enables to efficiently determine which tile points are actually contained in the footprint. Efficiency is achieved by pruning off large sets of the points corresponding to branches of the octree close to its root. The remaining points are then tested, as described in Section 3.3, to determine if they are required for the terrain approximation and rendering.

3.3 Continuous Resolution

Each DTM point has a grade quantifying its importance in the terrain approximation. This grade is traded off with distance from the viewpoint. In other words, more distant points are considered less significant. In practice, the client considers a virtual cone centered at the viewpoint, and calculates which DTM points in the geometry cache have a grade positioning them *inside* the cone (see Fig. 3). We would like to be able to determine this set of points in time proportional mainly to their number (and not to the total number of points in the viewing frustum). In computational-geometric terminology, this is called *output-sensitive range reporting*. We achieve this again using the tile octree. The complexity of the range reporting procedure is $O(\sqrt{N}+k)$, where N is the number of points in the viewing frustum, and k the number of points in the answer to the query ([13], p.79). Using this virtual cone also implies that a small change in the viewpoint induces a small change in the DTM points used for the rendering, thus ensuring the temporal continuity of the rendered images.

3.4 Caching

Portions of geometry tiles are imported from the server on demand and stored in the client cache. Only the neccesary upper levels of the tile octree are imported, possible due to the fixed structure of the octree. Hence a typical snapshot of the client cache contents would reveal a few (foreground) tiles from which almost the entire data content has been read, and many (background) tiles with a very sparse content. A prediction mechanism, based on the viewpoint trajectory, enables the loading of tiles in advance, resulting in smooth streaming of geometry from server to client.

3.5 Dynamic Delaunay Triangulation

The piecewise linear surface induced by the *Delaunay triangulation* of the 2D projection of the DTM points is generally considered the most suitable for surface approximation. This is because the minimal angle in the triangulation is maximized, eliminating long "slivery" triangles. Hence, the client constantly maintains a Delaunay triangulation of the DTM points contributing to the approximation of the terrain in the footprint. Many $O(n \log n)$ time algorithms exist for the Delaunay triangulation of n points, but not many are able to efficiently support update of the triangulation upon insertion or deletion of points. We use the algorithm of DeVillers et al [7], which inserts points in $O(\log n)$ and deletes points in $O(\log \log n)$ average time using a hierarchical data structure. Care must be taken to slightly perturb the spatial positions of the DTM points, otherwise degeneracies in the Delaunay triangulation and unstable numerics may occur.

At the client, points which were in the footprint corresponding to the previous frame, and are no longer in the current footprint, are removed from the triangulation - the main geometric data structure maintained online by the client. New points which were previously not in the footprint, and now are, are inserted into the triangulation. The turnover of points in the triangulation depends on the viewpoint velocity. Theoretically, very large velocities could cause successive frames to see totally different regions of the terrain, requiring the formation of an entirely different triangulation between frames. In practice, however, this does not occur. Typically, 99% of the footprint areas overlap between successive frames.

Pseudo-code of the flow of control in the client while rendering a single frame appears in Fig. 1.

4 Texture Processing

The texture data must also be manipulated at multiple resolutions, since image foreground pixels contain high resolution texels, and image background pixels contain low resolution texels. The resolution of the texels contributing to any given image pixel is essentially a function of the viewing distance to that scene point. The server texture database is also organized in tiles, storing the texels compressed to approximately 30% of their original volume, using a progressive wavelet scheme. This results in a bit stream sorted by importance.

A typical low-end client computer contains a texture buffer of limited capacity (e.g. 1024x1024 pixels) with a pyramid structure on top of it. By supplying appropriate texture coordinates for the rendered triangle vertices, the graphics hardware/software maps texels from the texture buffer to the image pixels in the interior of the projected triangles. Each level of the texture pyramid contains texels representing the same terrain area, at decreasing resolutions. However, since not all texels, especially not at *all* resolutions, will contribute to the terrain image (see Fig. 4), there is no need to import them from the server. We optimize network bandwidth by loading *only* those texture tiles which intersect the view footprint, at the appropriate resolution, if they are not yet loaded. By this we mean we calculate the number of encoded bits of the texture stream required to reconstruct the texture tile at the appropriate res-

gons in the approximation is more or less constant, independent of the viewing parameters (for a fixed frame rate). For the texture,we employ a progressive wavelet compression scheme [2], which enables the extraction of texture at a continuum of resolutions from arbitrary prefixes of the encoded bit stream.

Our ultimate goal is to render any terrain image in time proportional to the *image* resolution (in pixels), and not to the scene complexity, number of DTM points in the viewing frustrum, texture resolution, etc. We are motivated by the (simple) observation that an image of fixed resolution can contain only a bounded amount of information, therefore any algorithm rendering such an image should not use more than a bounded number of polygons and texels. Such algorithms are called *output-sensitive*. Most algorithms are not output-sensitive, and in order that they be such, require careful design. Our system contains a careful blend of techniques, some borrowed from computational geometry, which together achieve a high degree of output sensitivity, enabling adequate performance in a limited-resource environment.

Since one server may be accessed simultaneously by a large number of clients, is is crucial to minimize the amount of work the server performs per client. If this load is minimized, the server will be scalable, able to support a virtually unlimited number of clients. We adhere to this principle throughout our implementation.

Using these methods, we have developed a client application achieving terrain visualization at interactive rates on a low-end SGI (O_2)workstation, accessing a server database over a network with bandwidth comparable to the Internet. This paper describes the architecture and algorithms incorporated into our system.

2 System Overview

The large terrain scene resides on the server disk, partitioned into geometry and texture *tiles* of fixed size. A raw geometry tile contains a matrix of elevation heights, and a texture tile a matrix of texels. Tiling schemes are standard in terrain visualization applications (e.g. [4]). The server processes requests for geometry and texture data received from remote clients. In a preprocessing step at the server, applied independently to each tile (thus enabling a scene consisting of an unlimited number of tiles), the DTM points are assigned "grades" related to their importance in approximating the terrain surface. These grades are obtained from the *simplification* algorithm of Heckbert and Garland [9]. Using these grades as a third dimension, the DTM points in each tile are organized into a 3D octree, which will enable efficient answers to future geometric queries. The client maintains online a geometry cache containing DTM points from a small subset of the server's geometry tiles. Even from these tiles, only the relevant upper levels of the corresponding octrees are imported to the client. Which levels are relevant is determined on the fly by the client.

At any given moment, a subset of the geometry cache points are maintained at the client in a dynamic Delaunay triangulation, our primary geometric data structure. To maintain the triangulation, we use the algorithms of Devillers, Meiser and Teillaud [7] for efficient insertion and deletion into a 2D Delaunay triangulation. Delaunay triangulations are commonly considered to be suitable for terrain visualization purposes. A DTM point deserves to be in the triangulation if its grade is greater than a threshold, which is proportional to the distance of the point from the viewpoint. Section 3 elaborates on the details of how we handle the geometry.

The texture data is maintained at the server in tiles, compressed using the progressive wavelet scheme of Buccigrossi and Simoncelli [2]. This scheme compresses the data to approximately 30% of its raw size with negligble loss, and, more important, allows the decoding of the texture data from any prefix of the bit stream. Naturally, using more bits will result in a higher quality result. Client requests for texture data at a given resolution result in the streaming of the prefix of minimal length sufficing for the required resolution. Section 4 describes our handling of the texture in more detail.

The client graphics pipeline, sometimes supported in hardware, is fed relevant triangles and texels. This pipeline takes care of the basic rendering operations, e.g. perspective projection, hidden surface elimination, and texture mapping. The main issues we address in our implementation are the minimization of data transmitted from the server to the client caches and subsequently fed to the graphics pipeline.

Typical triangulations and rendered images generated by our client system are shown in Fig. 2.

3 Geometry Processing

3.1 Data Reduction

A typical DTM is supplied on a regular grid, and this data is usually highly redundant. If the surface is to be approximated by a piecewise-linear 2D function (a collection of planar polygons), a small number of large polygons suffice to approximate the surface well in planar regions. On the other hand, terrain areas with high curvature, such as ridges and ravines, require a large number of small polygons to achieve a satisfactory approximation (see Fig. 2). By this argument, is it obvious that some DTM points are more important than others. Heckbert and Garland [9] have described a procedure which starts off with a small number of DTM points (usually the four corners of the DTM coverage), and incrementally adds points whose contribution to the surface approximation is most significant. The contribution of a point to the approximation is quantified by its vertical distance from the piecewise-linear approximation built with all previous points. The larger this distance - the more important the point is. The incremental procedure is done efficiently using a priority queue mechanism.

We use the Heckbert and Garland procedure at the server as a preprocessing operation on each tile to assign each DTM point a numeric "grade" - precisely the vertical distance described in the previous paragraph. This grade is stored with the point, and used later to determine online whether the point is required for the terrain approximation. This decision is based on the grade and the point's distance from the viewpoint. To facilitate efficient decision-making, we build a 3D octree of the DTM points, the grade serving as the third dimension. The grid structure of the points in the XY plane facilitates a fixed quadtree structure in this plane, which, in turn, facilitates the organization of the data stored in the tile in a

Visualization of Large Terrains in Resource-Limited Computing Environments

Boris Rabinovich Craig Gotsman

Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
[borisr|gotsman]@cs.technion.ac.il

Abstract

We describe a software system supporting interactive visualization of large terrains in a resource-limited environment, i.e. a low-end client computer accessing a large terrain database server through a low-bandwidth network. By "large", we mean that the size of the terrain database is orders of magnitude larger than the computer RAM. Superior performance is achieved by manipulating both geometric and texture data at a continuum of resolutions, and, at any given moment, using the best resolution dictated by the CPU and bandwidth constraints. The geometry is maintained as a Delaunay triangulation of a dynamic subset of the terrain data points, and the texture compressed by a progressive wavelet scheme.

A careful blend of algorithmic techniques enables our system to achieve superior rendering performance on a low-end computer by optimizing the number of polygons and texture pixels sent to the graphics pipeline. It guarantees a frame rate depending *only* on the size and quality of the rendered image, independent of the viewing parameters and scene database size. An efficient paging scheme minimizes data I/O, thus enabling the use of our system in a low-bandwidth client/server data-streaming scenario, such as on the Internet.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; D.4.4 [Operating Systems]: Communications Management—Network Communication.
Keywords: Terrain rendering, level-of-detail, interactive graphics

1 Introduction

Terrain visualization is an important component of many civilian and military applications [10, 3]. The input to the terrain visualization problem is usually a large Digital Terrain Map (DTM), consisting of elevation data sampled on a regular grid, and corresponding aerial and/or satellite texture data, which is mapped onto the reconstructed terrain surface. The output is rendered images of the terrain surface, usually as part of a "flythrough" sequence.

The advent of the World-Wide-Web suggests the running of this type of application over the Internet, in a client/server scenario. The server is a very large remote database, accessed by the client, usually a low-end computer, over a narrow-bandwidth line (3 KByte/sec is typical for the contemporary Internet). The two bottlenecks that have to be overcome are the bandwidth in delivering relevant terrain data from the server to the client, and the CPU power required at the client for rendering this data.

The key to efficient terrain *rendering* is efficient online manipulation of both the geometric and texture data, especially when the scene database at the server is orders of magnitude larger that the size of client system RAM. Naive terrain rendering algorithms convert each DTM cell (bounded by four adjacent grid points) into two 3D triangles, and render (send through the graphics pipeline) all such triangles in a region determined by the viewing frustum. They also map the texture data at its highest resolution onto these polygons. This is a very inefficient procedure, as for low pitch angles, the number of these triangles and texture pixels (*texels*) may be extremely large. Each individual triangle projection to image space is very small, and many texels may be condensed to one image pixel, contributing negligibly to the image. One remedy to this problem, adopted in a number of works over the past few years (e.g. [8]) is to maintain the scene data at a number of discrete *levels-of-detail*. Since terrain areas at large viewing distances project to small image areas, there is no point rendering them in full detail. At any given moment during the animation, the appropriate level-of-detail is used to render the image. To do this effectively, pieces of the scene must be taken from multiple levels (foreground areas from a high-detail version, and background areas from a low-detail version), requiring methods to "stitch" together pieces of different models in a continuous fashion, so that there are no holes or breaks along the seams. This has proven to be a major problem for the geometric data, since there usually is no topological correlation between the different levels of detail. De Berg and Dobrint [1], Cohen-Or and Levanoni [5], and Lindstrom et al. [12] have provided partial solutions to the stitching problem.

In this paper we use a different approach to maintaining the terrain geometry, proposed independently by Klein and Huttner [11] and Delepine [6]. The geometry is treated in a *continuous-resolution* fashion. We do not maintain multiple geometric models (at different levels of detail), rather continuously update one model online to represent in an optimal way the projection of the terrain contained in the viewing frustum. As a result, the number of poly-