Real-Time Terrain Visualization on PC

Jan Vaněk
Faculty of Informatics and Management
University of Hradec Králové
Víta Nejedlého 573
500 03, Hradec Králové, Czech Republic
janvan@post.cz

Bruno Ježek
Purkyně Military Medical Academy
PO.Box 35
Třebešská 1575
500 01, Hradec Králové, Czech Republic
jezek@pmfhk.cz

ABSTRACT

Terrain visualization is a task taking place in all geographic information systems (GIS). Very large datasets are often processed and visualized in real-time. Although the performance of hardware continually increases, it is necessary to optimize the form and the amount of 3D terrain information to obtain sufficient framerates during visualization. This article deals with choosing the appropriate terrain representation and a suitable method for large terrain data management. Proposed optimizations were successfully tested in implementation of real-time terrain visualization using a common graphic card for hardware acceleration.

Keywords

terrain visualization, real-time rendering, hardware acceleration, geographic information systems, level of detail

1. INTRODUCTION

As geographic information systems (GIS) are being used widely by many business and public institutions for a great variety of purposes, it is becoming quite urgent to extend visualization techniques employed in existing GIS. Although commonly used two-dimensional maps carry large amounts of information, the actual terrain shape is not evident. Reading the shape from contour lines and using it to make qualified decisions requires several years of practice.

Three-dimensional visualization of geographical data provides information about the shape of the terrain and locations of other objects on a map quickly and easily. An interactive system with real-time 3D terrain rendering is much more useful for users.

At the beginning it is advisable to specify the requirements and restrictions of the software for terrain visualization:

1. The rendering must be done in real-time. Only framerates above 15 frames per second are considered to be sufficient.
2. The software must handle terrain hundreds of kilometers in size with a detail of ten or less meters.
3. These goals must be met on consumer-level hardware. Expensive professional hardware must not be required. On the other hand, the software should take advantage of what modern hardware has to offer.

2. TERRAIN REPRESENTATION

One of the first methods of terrain rendering – voxel-based terrain rendering [Kau99] – is an extension of volume data rendering. Although successful real-time voxel-based terrain rendering engines exist [Outcast], this terrain representation is not suitable for our purpose because there is no consumer hardware capable of real-time volume data rendering.

Within other representations the only truly real-time renderable scene representation is a triangulated network, which is for many reasons perfectly suitable for terrain visualizing. It can be simply rendered using z-buffer which is hardware accelerated by the vast majority of graphic cards available to a common consumer.

Even with a triangulated network as a representation, we have some options to choose from. A triangulated irregular network (TIN) can be used to

WSCG POSTERS proceedings
WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press

represent any arbitrary shape including cliffs or overhangs but geometries of such formations are not stored in GIS databases and because of the scale of the considered terrain these formations can be ignored.

A regular network – a triangulated height-map for example – is another option. We have decided to represent the terrain by a height-map because of its several advantages. A decrease in the storage capacity required for saving terrains is one of the most important. If a triangulated height-map of 4096×4096 points is stored in the form of a TIN with 32-bit precision and with the most universal triangle description (3 indices per each triangle), it will require at least 575MB instead of 64MB for the height-map itself (32-bit precision as well).

Another advantage of height-maps comes from modern methods of acquiring geographic data. Satellite and aerial measurement systems produce height-maps and if such data is not available it will be possible to reconstruct the height-map from contour lines [Gol02].

Representing the terrain as a height-map simplifies the detection of collisions between moving objects (including camera) and the terrain. For every rendered frame each object travels into a new position. The software detects collision by comparing the altitude of an object and the altitude interpolated from the height-map and this is done with constant complexity (i.e. in O(1)).

Height-Map Triangulation

In order to render a terrain in real-time with a sufficient framerate we are using hardware acceleration via Direct3D and therefore we must compile the terrain representation into compatible data structures – vertex and index buffers.

A vertex buffer is an array of structures (records) describing each vertex. A structure sufficient for our purpose consists of three 32-bit floating point numbers for the vertex position and another three for the vertex normal.

An index buffer describes how vertices shape triangles. It contains indices pointing to the vertex buffer. Indices in an index buffer can be organized in several schemes: a triangle list, a triangle fan or a triangle strip.

A triangle list is universal as it may describe any ordering of triangles. Each triangle is represented by three indices. A triangle fan is more efficient considering the number of indices but it requires a certain organization of the triangles. All the triangles must have one vertex in common; each triangle is adjacent to two triangles and the adjacency edges must contain the joint vertex. The first index in the index buffer points to the joint vertex; any other two indices specify a triangle.

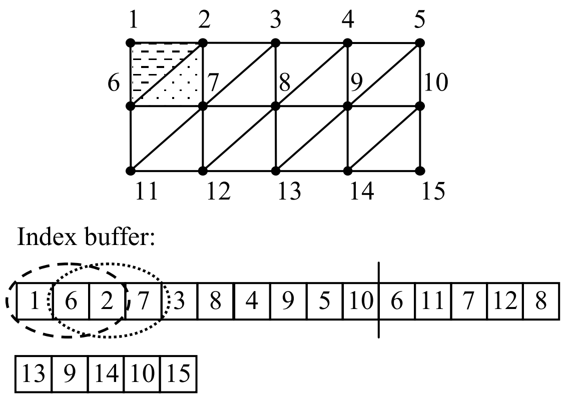
A height-map triangulation cannot be described as one triangle fan and therefore the index buffer must be split into parts (one part per each triangle fan) that are processed individually. This creates overhead expenses because in Direct3D 8 a function must be called to render each such part. Under OpenGL creating a display list hides the overhead (such as additional data structures management) inside the OpenGL implementation. The number of triangle fans covering a $m \times n$ height-map is $\frac{1}{4} \cdot (n - 1) \cdot (m - 1)$. To triangulate a whole height-map with triangle fans, both m and n must be odd.

A triangle strip is even more efficient in terms of the number of indices and definitely more efficient considering the number of parts. Any three successive indices in the index buffer describe one triangle; to add a triangle to the strip one additional index is required (see Figure 1). The index buffer is broken into $m - 1$ parts and thus the overhead costs are significantly lower than in the case of a triangle fan, but they are still present.

All the overhead, which comes from processing an index buffer split into parts, must be handled by the CPU. Since a height-map can't be efficiently rendered without the overhead costs of using triangle fans or triangle strips, we came up with an idea of replacing the overhead with another form of additional computational expenses.

Triangulation description	Number of indices	Number of parts	$m = n = 1025$	
			Index buffer size	Number of parts
Triangle list	$6 \cdot (n - 1) \cdot (m - 1)$	1	24 MB	1
Triangle fan	$\frac{5}{2} \cdot (n - 1) \cdot (m - 1)$	$\frac{1}{4} \cdot (n - 1) \cdot (m - 1)$	10 MB	262144
Triangle strip	$2 \cdot n \cdot (m - 1)$	$m - 1$	8.008 MB	1024
Degener. triangles	$(2 \cdot n + 2) \cdot (m - 1)$	1	8.016 MB	1

Table 1: Overview of triangulation descriptions



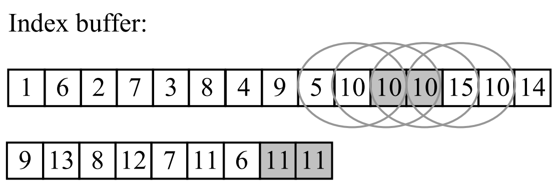
Number of indices: $2 \cdot n \cdot (m - 1)$

Figure 1: Triangle strip

We have extended the triangle strip so that the whole height-map triangulation can be described as one triangle strip. It can be achieved by adding four triangles (two indices) to each row. These triangles are degenerate as at least two of their indices point to one vertex and therefore they are not visible. This way the additional CPU load was replaced by a lighter additional GPU load. A well designed GPU should recognize a degenerate triangle and skip it. So the overhead of adding degenerate triangles is minimal.

Figure 2 shows the added indices and triangles. One index (and thus one triangle) is appended in order to maintain a correct vertex orientation. Otherwise the backface culling would remove the odd rows. With the backface culling switched off only one additional index and three triangles are needed.

As we can see in Table 1 the triangle strip with degenerate triangles is the most efficient height-map triangulation description. During our tests on a 1GHz machine equipped with GeForce4 Ti 4400, switching from the triangle strip to the triangle strip with degenerate triangles doubled the framerate on some terrains; all tests showed an increase in the framerate.



Number of indices: $(2 \cdot n + 2) \cdot (m - 1)$

Figure 2: Triangle strip with degenerate triangles

Let the height-map have m rows and n columns. The vertex indices then range from 0 to $m \cdot n - 1$. Index buffer construction is described by the following code fragment:

```
for (int i=0; i<m-1; i++) {
   if (i%2) {
      for (int j=n-1; j>=0; j--) {
         add_index((i+1)*n+j);
         add_index(i*n+j);
      }
      add_index((i+1)*n);
      add_index((i+1)*n);
   }
   else {
      for (int j=0; j<n; j++) {
         add_index(i*n+j);
         add_index((i+1)*n+j);
      }
      add_index((i+2)*n-1);
      add_index((i+2)*n-1);
   }
}
```

The number of indices in the index buffer is $(2 \cdot n + 2) \cdot (m - 1)$ which equals the number of executions of the subroutine add_index.

3. LARGE HEIGHT-MAP

An optimal terrain representation itself does not suffice for real-time rendering. A triangulation of a height-map of $10\,000^2$ elements (a 100^2 km terrain with a 10 m detail) consists of around 200 million triangles and in the most compact renderable form described above, it takes up over 3 GB of memory (of the graphics card). To maintain a framerate over 15 fps the GPU would have to process more than 3 billion triangles per second. Such a hardware configuration is not likely to appear in the foreseeable future. On top of that, the requirements are growing more rapidly then the hardware performance. Therefore we have to reduce the number of displayed triangles.

The most complex and general algorithms of triangulation reduction (e.g. [Pup96]) are not suitable for hardware accelerated rendering. These algorithms work with individual vertices, edges and triangles. Therefore the geometry is often changed and these changes must be transferred from the system memory to the memory of the graphics card.

Reducing the complexity of a larger group of triangles is more suitable for hardware accelerated rendering as the change of geometry is made less often. Rules for deciding whether to remove a vertex or not are not applied on every vertex but on a relatively large group of vertices and thus CPU requirements are many times lower.

A height-map representation has another advantage – it can be simply divided into overlapping patches. Each patch covers $2^n + 1 \times 2^n + 1$ vertices so that it is easy to reduce the number of triangles by removing odd vertices (see Figure 3). The height-map is split into $r \times s$ patches and thus its size must be $r \cdot 2^n + 1 \times s \cdot 2^n + 1$ or else it must be resampled.

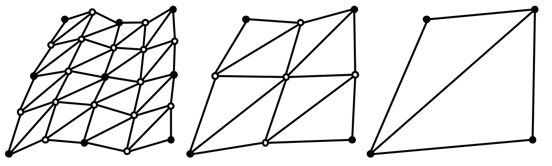


Figure 3: Patch level of detail

The result of removing all odd vertices from a patch of $2^n + 1 \times 2^n + 1$ vertices is a patch of $2^{n-1} + 1 \times 2^{n-1} + 1$ vertices. This patch can be reduced exactly the same way and so on down to a patch of 2×2 vertices. So a $2^n + 1 \times 2^n + 1$ patch can be rendered in $n + 1$ levels of detail – patches of $2^k + 1 \times 2^k + 1$ vertices; $k = 0,\ldots,n$ specifies the level of detail (LOD) of the patch.

LOD is selected depending on the patch distance from the camera position. If *dist* is less than *mindist* then $k = n$, else if it is greater than *maxdist* then the patch is at level 0, else the LOD is given by the following expression: $k = n \cdot \left(1 - \left(\frac{dist - mindist}{maxdist - mindist}\right)^{exp}\right)$. By specifying *mindist*, *maxdist* and *exp* a user can set the overall scene detail.

Cracks occur on borders of adjacent patches of different levels of detail. To remove them the geometry must be changed. The index buffer for each LOD is precomputed and thus all the patches use the same index buffer per LOD. This improves performance. Therefore the cracks are removed by modifying the vertex buffer. Vertices of the patch of the greater LOD that do not match vertices in the patch of the lower LOD are moved to the vertices that match (see Figure 4).

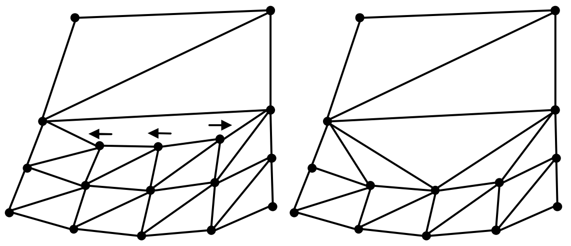


Figure 4: Crack removal

View-frustum culling is another method for reducing the number of displayed triangles. The simplest way to implement it is to create an axis-aligned bounding box (AABB) for the patch, project the AABB to the screen space and then check whether the AABB intersects the view-port rectangle.

4. RESULTS

All the proposed solutions were implemented and successfully tested in a real life application employed at the Purkyně Military Medical Academy. During preparations for field training, students have used GIS in the common two-dimensional form and faced the problem of recognizing the terrain shape. The request for 3D visualization was quite obvious.

We have tested different height-map resolutions from 1025^2 up to 20481^2 vertices on several computers ranging from a 1GHz machine with nVidia GeForce4 Ti to a 3GHz one equipped with ATI Radeon 9800. For each map and hardware configuration we were able to set the LOD parameters so that the average framerate was above 15 fps and the overall detail still made a good impression. A height-map of 20481^2 vertices (split into 80^2 patches of 257^2 vertices) was rendered with an average of around 850 000 triangles per frame to keep the scene in a sufficient detail.

5. CONCLUSION

Solutions of several problems concerning real-time terrain visualization on common hardware accelerated graphic cards are described in this work. As an optimized terrain representation we have designed the extended triangle strip with degenerate triangles, which is the most efficient height-map triangulation description.

A method for dividing the whole terrain into regular patches with different levels of detail is proposed for large height-maps. The problem of connecting the patches is solved.

All the proposed optimizations were implemented and tested on several computers with common hardware configurations.

6. ACKNOWLEDGMENTS

The authors would like to thank the Graphics Group of the University of Hradec Králové, especially Associate Professor Antonín Slabý for funding this work.

7. REFERENCES

[Gol02] Dakowicz, M., Gold, C. Visualizing Terrain Models from Contours - Plausible Ridge, Valley and Slope Estimation. Proc. International Workshop on Visualization and Animation of Landscape, 2002.

[Kau99] Kaufman, A., Qu, H. and Wan, M. Virtual Flythrough Over Voxel-Based Terrain. Proc. IEEE Virtual Reality '99, pp. 53-60, 1999.

[Outcast] Engine for the computer game Outcast developed by Appeal, published by Infogrames, http://www.outcast-thegame.com, 1999.

[Pup96] Puppo, E. Variable Resolution of Terrain Surfaces. Proc. Eighth Canadian Conference on Computational Geometry, 1996.