EARTH 3D MODELING FOR WEATHER FORECAST PRESENTATION

Imed Jabri and Tahar Battikh

Ass. Professors Electrical engineering Dept. ESSTT
Tunis, TUNISIA, Email : imedjabri@yahoo.com and btahar@yahoo.com

Abstract: *In this paper, we put forward the development of a solution allowing modeling and exploitation of real time synthesized images of the globe. This is used for presenting the weather forecast bulletin of Tunisian television Channel 7. This presentation requires a smooth panning to any part of the world, while having the necessary level of detail. To store and handle geographical data, most of the used methods implement a tree structure of data where various threads come out of each node (four in the case of quadtree). Recursively, one goes down the tree until a satisfactory level of detail is reached. Our contribution consists in the adaptation of this algorithm to a sphere and the exploitation of various filtering, storage and optimization techniques in connection with the available material*

Keywords: *visualization, multi-resolution geometrical modeling, rendering based on view parameters, extra main memory rendering, level of detail, geomorphing, texture mapping, fuzzy controller.*

1. INTRODUCTION

In order to control the events which depend on the weather conditions (agriculture, aviation, tourism...), the user becomes increasingly demanding concerning the quality of the synthesis of geographical information and its dynamics. The technological development (satellite transmission, graphics boards, processor, memory size...) offers today a considerable mass of data in real time related to space.

The globe representation to which we associate useful information (weather, demography, economy...) must be as faithful and practical as possible exploiting middle ranged PCs and graphic cards.

We recommend illustrating the terrestrial sphere in navigation in real time for the presentation of a weather forecast bulletin (ground, sea level, sky, clouds, rain, snow, fog, weather symbols) with a level of detail, image and elevation, going up to 250 m/pixel for Tunisia and 1km/pixel for the rest of the world. Because of the great mass of raw data related to the 3D representation of the parameters of the globe (texture and elevation), their storage requires a great capacity of 4.618 Gb (for a resolution of 1km/pixel) distributed as follows:

Texture: 43800 x 21600 x 3 (RGB) = 2.771 Go
Elevation: 43800x21600 x 2 (16bits) = 1.847Go

For the elevation presentation with a variable level of detail "LOD" and with a realistic appearance, it is necessary to take into account the real time constraints.

As PCs and graphic processors present limits in terms of size of the data and vertex and that the access disk should not significantly block the rendering speed, it is necessary to carry out an optimization of the useful information display, and judicious storage of the data.

2. PROCEDURE

The followed procedure consists in adopting an algorithm of a spherical field rendering then ensuring the filtering of the branches of the tree structure.

2.1 Algorithm of spherical field rendering

The parts of the field which are not close to the camera do not necessitate the same detail level as the near ones. They can be treated using a lower resolution in order to significantly reduce the number of vertex and thus increase the rendering speed. We have exploited this technique “LOD” in order to reduce the vertex number to submit to the graphic card and which consists in dynamically subdividing the field into hierarchical detail levels [P[2]].

2.2 Adaptation of the Geomipmapping algorithm on a sphere

In order to represent the earth using a variable detail level, we adopted the geomipmapping algorithm (traditionally applied on a flat surface) on a sphere. Hence, we exploit the spherical data (ρ, Θ, φ) instead of the Euclidian ones (x, y, z). The subdivision recursion makes us use a square data matrix (2^nx2^n). The spherical data (φ : -180…+180 et θ : -90..+90) are shown in a rectangular matrix 2 x (2^nx2^n), hence the idea of exploiting each half of the sphere separately.[18]

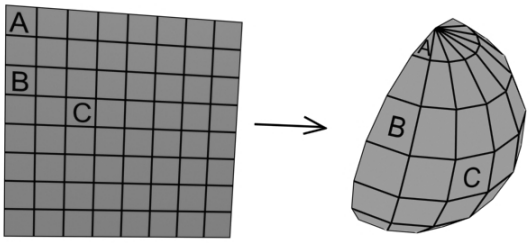

Fig. 1. Correspondence between a square surface and a half of sphere

The iterations enable us to model the globe having an increasing number of sides ($2x4^1$, $2x4^2$, … $2x4^n$)

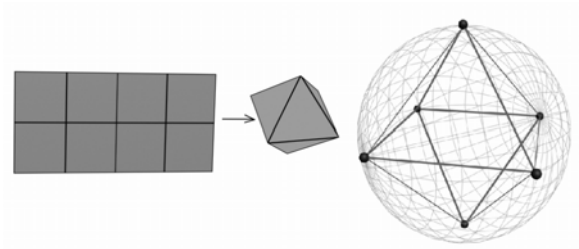

Fig. 2. Iteration 1: Subdivision of the sphere into 8 sides

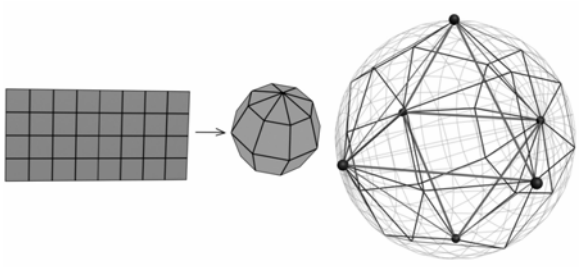

Fig. 3. Iteration 2: Subdivision of the sphere into 32 sides

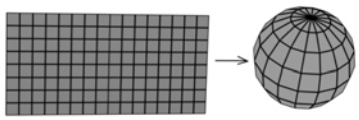

Fig. 4. Iteration 3: Subdivision of the sphere into 128 sides

2.3 Filtering the invisible branches of the tree structure

2.3.1 Filtering per frustum culling

When an important part of the globe lies outside camera range, it is filtered through the use of the frustum culling technique which enables us to

rapidly eliminate patches that will not be displayed in the final rendering of the visible part of the globe. Thus, it avoids calculating useless data and sending them to the graphic card, and thus saves CPU and GPU resources as well as the memory bandwidth.

Frustum culling necessitates placing the objects of the scene in a hierarchy. If it is decided that an element of high hierarchy can be ignored, testing and rendering all the subordinate elements are avoided.

The quadtree approach consists in dividing the scene into four squares (4 childs) and then proceeds by considering each one of them as a parent. If one does not belong to the frustum, it's ignored; otherwise its four squares are subdivided again, and so on.

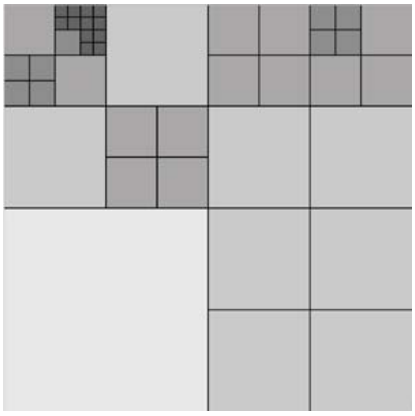

Fig. 5. Quadtree: spatial hierarchy

2.3.2 Filtering per culling sides

In a sphere within the camera range, sides whose normal vectors are in the same direction as those of the camera are invisible. As a consequence, they are ignored just like the ones that are outside the camera range.

This approach is based on a simple observation: if we know the normal of a side n, and the view direction v of the observer, it is possible to determine if this side would be seen or not. In fact, any side whose normal points to an opposite direction would be obscured. Mathematically, if the normal of a side makes an angle of less than 90° in the view direction, the side will then be dissimulated. In order to detect such a configuration, one has to realize the dot product between n and v.

In fact, $n \bullet v = ||n|| \bullet ||v|| \bullet \cos(\theta)$, θ measuring the angle between n and v. the sign of cos(θ) determines the sign of the dot product. If this angle is greater than 90°, then cos(θ) is negative, otherwise, it is positive. A negative dot product means that we can see the side, whereas a positive one indicates that it is obscured.[11]P

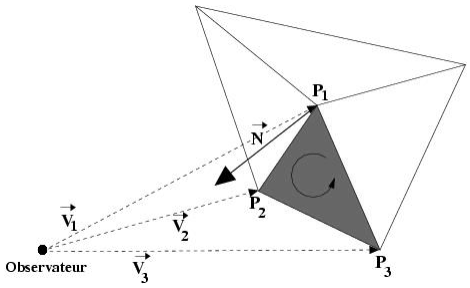

Fig. 6. Calculating the normal of a side

2.4 Subdivision and choice of a higher detail level

At a certain detail level, the use of the quadtree techniques require a too long computing time. As a consequence, we have to stop the subdivision at the 8th iteration that lets out 512x256 patches. The width of the initial texture is 43800 pixels, the closest power of 2 value is 65536, and the highest level of detail of each patch is 128x128 pixels (65536:512=128). If the obtained detail level is unsatisfactory, we subdivide this portion into a maximum of 128 equal parts, thus avoiding the interpolation of intermediary values and obtaining a visible satisfactory result.**P** [11]

At the beginning, we have to determine the desired detail constant (nd), once the visible zone is determined, the tessellation of each patch must not be the same. The level of detail should thus be proportional to camera distance. The patch is then divided by 2^n (n=0..7)

The following diagram illustrates the nd value with respect to the eye position. (0: maximum detail level)

Fig. 7. Determination of the desired detail constant (nd)

Close to the poles, the number of patches increases significantly, hence the necessity to multiply the length of the arc by (Cos **θ**) (-90 < **θ** < +90) to rapidly help converge the algorithm and present patches with a reasonable level of detail**.**

2.5 The algorithm

Render_Patch (θ, φ, delta(θ, φ), Level)
//frustum culling
If all the patch corners are outside the camera frustum **then**
Destroy childrens and exit
End If

//face culling
If level >3 and visibility_face_test=false then
Destroy childrens and exit
End If

//subdivision
If level<stoplevel then
Render_Patch (θ , φ , delta(θ, φ)/2, level+1)
Render_Patch (θ+plage/2 , φ , delta(θ, φ)/2, level +1)
Render_Patch (θ+plage/2 , φ+plage/2 , delta(θ, φ)/2, level+1)
Render_Patch (θ , φ+plage/2 , delta(θ, φ)/2, level+1)
End If

//choise of detail level
Detail=0
Perim=(EarthRadius * Cos(θ + delta(θ, φ) /2)*2*pi/(2 ^{level}))/ cameraDistance
While (Perim/(2 ^{Detail}) > nd) and (Detail < 7) DO
Detail := Detail+ 1
While End

// childs Render
- Search of texture and elevation of the child in the disk
- Mesh creation (Grid of 2^{level} x 2 ^{level})
- Submission to graphic board

3. PROBLEMS AND SOLUTIONS

3.1 Optimization and data saving

3.1.1 Optimization the data size

In order to optimize the amount of data stocked on disk, we have adopted various techniques :

For the texture: 128x128 pixel textures are compressed into DDS format (which is decompressed in real time by the graphic card), then DDS files are recompressed by Huffman algorithm compression, from the memory.

This technique allows us to obtain a compression ratio of about 1 : 6.

For the elevation: In addition to using Huffman algorithm compression, we exploit the fact that nearly ¾ of the globe is covered by water (elevation 0); as a consequence, we only note the dry land (elevation>0)

The following table contains texture size and elevation for the different detail levels.

Levels	Width	Height	Texture size	Elevation size	Total Elevation+texture
Level 8	65536	32768	6442450944	4294967296	10737418240
Level 7	32768	16384	1610612736	1073741824	2684354560
Level 6	16384	8192	805306368	536870912	1342177280
Level 5	8192	4096	402653184	268435456	671088640
Level 4	4096	2048	201326592	134217728	335544320
Level 3	2048	1024	100663296	67108864	167772160
Level 2	1024	512	50331648	33554432	83886080
Level 1	512	256	25165824	16777216	41943040
TOTAL					**16064184320**

Fig. 8. Texture size and elevation for the different detail levels

The different compression techniques allowed reducing the 16 GB space into 1 GB on the disk.

3.1.2 Data saving technique

In order to optimize disk access time, at a given level, we do not save the data blocks sequentially. In fact, we save according to the camera frustum and their close neighbors.

These data have to remain in the cache memory. That is why we did not choose a sequential storage, of all the data, but only of the nxn blocks. After various attempts, we have adopted the values: n=8 for level 8, 4 for 7, 2 for 6 and n=1 for the remaining levels.

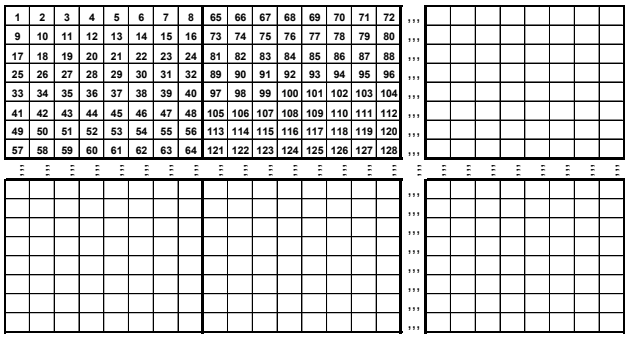

Fig. 9. Matrix of data saving technique

The algorithm used for data saving technique is as follow:

$a_{i,j} = j + (i-1).2^k + (p-1).2^{2k} + (q-1).(n+2^k-1+2^k(n-1)).2^k$

Where

$(q-1).2^k + 1 \leq i \leq q.2^k$ and $(p-1).2^k+1 \leq j \leq p.2^k$

$1 \leq p , q \leq n$

3.2 Subdivision number regulation

In order to maintain the image display rate at a constant value of approximately 25 images per second, we work on the parameters of detail level Nd.

The capacity of the exploited tools enables reaching a number of vertex (Nvtx) equaling 80000 ± ε%. The regulation consists in modifying the value of Nd so that we converge as fast as possible to the desired Nvtx. For this goal we used a fuzzy controller.

Traditional control systems are based on mathematical models in which the control system is described using one or more differential equations that define the system response to its inputs. Such systems are often implemented as "proportional-integral-derivative (PID)" controllers. They are the products of decades of development and theoretical analysis, and are highly effective.

In our case, the mathematical model of the control process is not obvious and may be too "expensive" in terms of computer processing power and memory, we believe that a system based on empirical rules should be more effective.

Furthermore, such systems can be easily upgraded by adding new rules to improve performance or add new features.

In order to control the value of Nvtx, we choose a fuzzy regulator of which here the diagram:

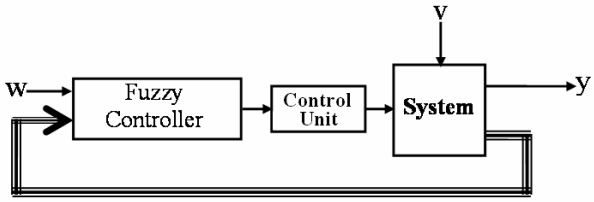

Fig.10. Fuzzy regulator diagram

The fuzzy controller developed consists of an input stage, a processing stage, and an output stage. The input stage defines the appropriate membership functions and truth values. The processing stage invokes rules and generates results, then combines the results of the rules. Finally, the output stage converts the combined result back into a control output value. [P5]

The shape adopted for membership function is triangular

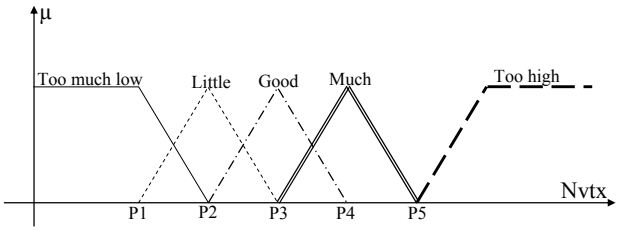

Fig. 11. Membership function

The control unit adopted has the following values and shapes.

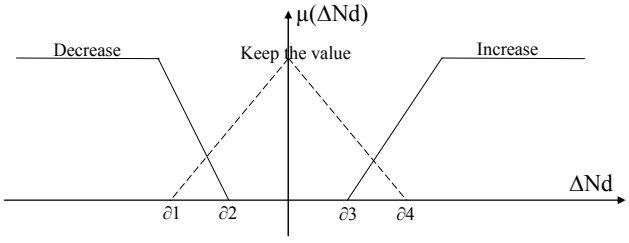

Fig. 12. Control output value

The fuzzy system developed has the following rule base:

- If Nvtx < Nvmax – ε then not enough vertices for the graphic board
- If Nvtx > Nvmax + ε then too many vertices for the graphic board
- If Nvtx = Nvmax ± α with α ≤ ε then we have a number of adequate vertex to the performances of our graphic board
- If Nvtx is definitely lower/higher than Nvmax then increase/ decrease appreciably Nd
- If Nvtx is slightly lower/higher than Nvmax then increase/ decrease moderately Nd

The following process chart positions the regulator in the solution suggested

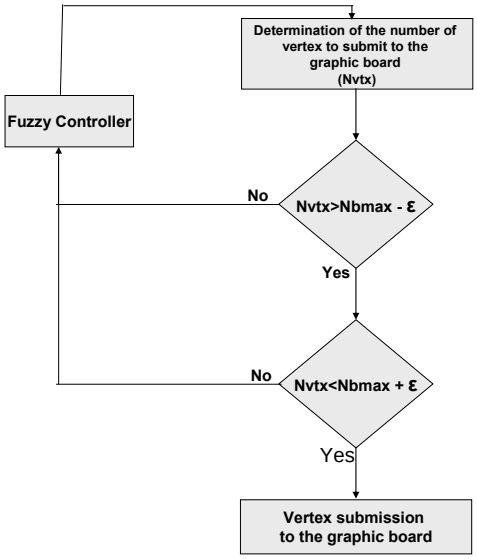

Fig. 13. Subdivision number regulation

After several trials the values adopted for P1, P2, P3, P4 and P5 are respectively 70000, 75000, 80000, 85000, 90000 and 0.85, 0.95, 1.05, 1.15 for δ1, δ2, δ3, δ4 respectively.

The fuzzy controller enabled us to maintain an fps number between 24 and 27 with an average equal to 25 fps at 96% of cases.

3.3 Smooth transition from a level to another

Transition from a level to another requires an access to disk that can slow down real time browsing. For this reason each time a different detail level is necessary, we cast a patch which prepares the desired detail level (lower or higher). After the execution of this task, rocking of the algorithm becomes possible.

4. OUTCOME AND COMMENTS

In the present work, we have used a Pentium 4.3Ghz PC with 1Gb RAM, 200GB Raid hard disk and an ATI Radeon 9600 Pro graphic card. We have used Delphi 6 and Windows XP.

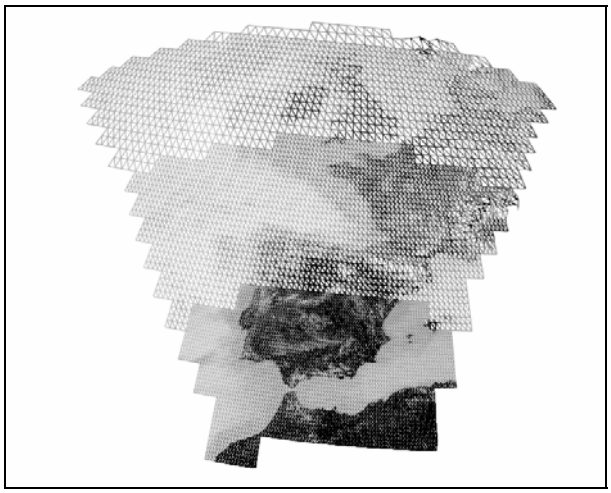

Fig. 14. Application of frustum culling

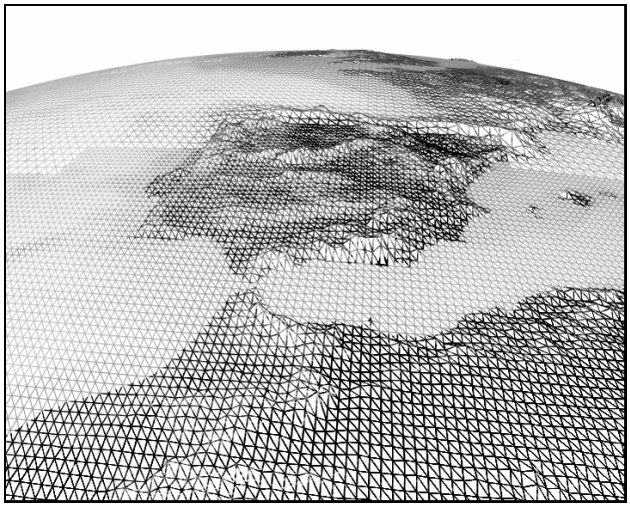

Fig. 15. Application of face culling

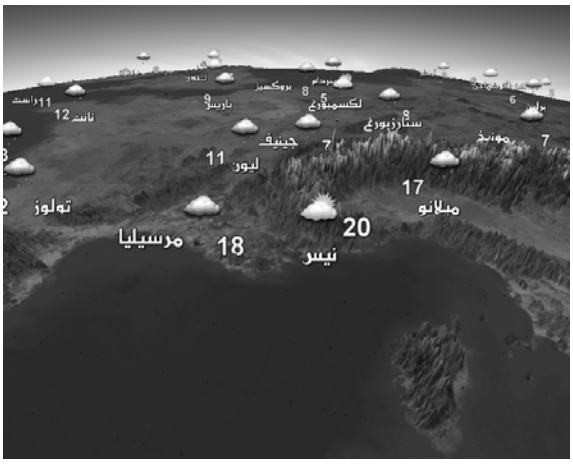

Fig. 16. Examples of final output

Each technique used contributed to the final solution which allowed a quasi-constant flow in the final output.

Techniques	Min fps	Max fps	Average fps
Basic Rendering	0	20	10
Adding disk saving techniques	10	25	12
Adding convergence at poles (cos multiply)	15	40	22
Adding fuzzy controller	24	27	25

Fig. 17. Contribution of the various techniques

5. PERSPECTIVES

5.1 Geomorphing

During the display, the abrupt transition from a detail level to another is not always quite smooth on the screen. If for a certain detail level, the successive elevation values are Z1 and Z2, the solution would be loading the data of the following elevation level into memory and continually moving to an intermediary level before applying on screen (Z' to (Z1+ Z2)/2 or from (Z1+ Z2)/2 to Z').

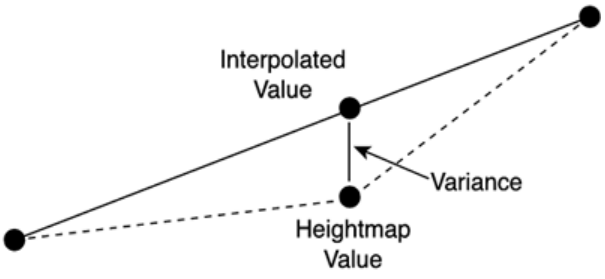

Fig. 18. Geomorphing technique

5.2 Exploitation of the vertex shaders and pixel shaders

We intend to exploit the possibilities of the new generation graphic cards which are less demanding in terms of processor resources and provide a better detail level. In fact, with the DX8, the vertex channel has become totally editable. Arbitrary vertex data are arbitrarily treated through the vertex shader who then writes the exit values in the registry in plain text**.**

The editable pixel channel has created for DirectX a totally new concept of pixel rasterization comparable to that of vertex shaders. Pixel shaders are going to revolutionize a domain in which everything depended on sophisticated, hindering practices of checking out the compatibilities with a unique function channel**.**

5.3 Pre-calculating the meshes

There is another idea which can contribute to the optimization of our solution which consists in pre-calculating the meshes, compressing and storing them in the space devoted to the elevations. This would reduce the computing time which is extremely precious and critical in real time browsing.

6. CONCLUSION

3D real time modeling of the globe in a 250m/pixel resolution is now possible with common PCs. This enables the free browsing of any corner of the globe with a realistic appearance (texture and embossment). The improvements that we wish to make accomplish with faster processors and more performing graphic cards would allow going further ahead with the data size that can be treated and the resolutions that can be reached.

REFERENCES

[1] BLOW, J. Terrain rendering at high levels of detail. In Game Developers Conference Proceedings (2000).

[2] CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. Planet-sized batched dynamic adaptive meshes (p-bdam). In VIS '03: Proceedings of the 14th IEEE Visualization 2003

(VIS'03) (Washington, DC, USA, 2003), IEEE Computer Society, p. 20.

[3] COHEN-OR, D., AND LEVANONI, Y. Temporal continuity of levels of detail in Delaunay triangulated terrain. In IEEE Visualization '96 Conference Proceedings, (1996) pp. 37–42.

[4] Cuevas, E.V., Zald´ıvar, D., Rojas, R. "Competitive Neural Networks Applied to Face Localization" Technical Report B-13-03, FU-Berlin, Germany, (2003).

[5] DUCHAINEAU, M., WOLINSKY, M., SIGETI, D., MILLER, M., ALDRICH, C., AND MINEEV-WEINSTEIN, M. ROAMing terrain: Real-time optimally adapting meshes. In IEEE Visualization '97 Conference Proceedings (1997), pp. 81–88.

[6] FAN, M., TANG, M., AND DONG, J. A review of real-time terrain rendering techniques. The 8th International Conference on Computer Supported Cooperative Work in Design Proceedings (2003), pp. 685–691.

[7] FERNANDO, R., AND KILGARD, M. The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Addison-Wesley Professional, 2003.

[8] GERASIMOV, P., FERNANDO, R., AND GREEN, S. Shader Model 3.0 Using Vertex Textures. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050 June 2004. Available at http://developer.nvidia.com/object/usingvertex textures.html.

[9] GREENE, N. Graphics Gems IV. Heckbert, 1994, ch. Detecting Intersection of a Rectangular Solid and a Convex Polyhedron, pp. 74–82.

[10] HAKL, H., AND ZIJL, L. V. Diamond terrain algorithm: Continuous levels of detail for height fields. South African Computer Journal (2002).

[11] HILL, D. An efficient, hardware-accelerated, level-of-detail rendering technique for large terrains. Master's thesis, University of Toronto, 2002.

[12] HOPPE, H. Smooth view-dependent level-of-detail control and its application to terrain rendering. IEEE Visualization '98 Conference Proceedings (1998), pp. 35–42.

[13] LEVENBERG, J. Fast view-dependent level-of-detail rendering using cached geometry. In IEEE Visualization '02 Conference Proceedings (2002)

[14] LINDSTROM, P., KOLLER, D., RIBARSKY, W., HODGES, L., FAUST, N., AND TURNER, G. Real-time, continuous level of detail rendering of height fields. In SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (1996), pp. 109–118.

[15] A. KANDEL, AND G. LANGHOLZ, ed., Fuzzy hardware, architectures and applications, Kluwer Academic Publishers, Boston, London, Dordrecht, 1998.

[16] LOSASSO, F., AND HOPPE, H. Geometry clipmaps: terrain rendering using nested regular grids. ACM Transactions on Graphics (2004), 769–776.

[17] MCNALLY, S. The Tread Marks engine. In Game Developers Conference Proceedings (2000).

[18] NVIDIA CORPORATION. Using Vertex Buffer Objects. 2701 San Tomas Expressway, Santa Clara, CA 95050, October 2003.

[19] PHARR, M., Ed. GPU Gems 2. Addison-Wesley, 2005.

[20] POLACK, T. Focus on 3D Terrain Programming. Premier Press, 2003.

[21] POMERANZ, A. ROAM using surface triangle clusters (RUSTiC). Master's thesis, University of California at Davis, 1998.