Parallel-Based Error Metric For Large-Scale Terrain Rendering Method

Zihou Ge[1, a], Xianyu Zhang[1]

[1] Aircraft and power department, Aeronautical University of Air Force, Changchun, 130022, China

[a]email: gezihou@163.com

Keywords: Parallel-based, Terrain rendering, Nested geometric space error

Abstract. This paper presents a terrain rendering technique for large, textured terrain, which uses Parallel-based error metric. In a preprocess, the domain was first tiled and computed into a nested geometric space error map based on geographic theory, taking the advantage to save displaying memory with terrain map. In order to obtain the seamless connection, the edge errors of two connecting tiles are consistent with each other. At run time, screen space error is projected from geometric error, and compare with error threshold according to view-point, to select terrain grid points by GPU. At last, triangle strip is constructed according to RQT method. The method processes the error metric and triangulation by GPU, to improve the rendering efficiency of image hardware, and save the running time of CPU.

Introduction

In this paper a new optimal design of soccer robot control system which is based on mechanical analyses and calculations on the pressure and transmutation states of chip kick mechanics, this new control system with high precision for speed control and high dynamic quality.

Large scale terrain rendering at real-time frame rates is an important area of computer graphics, which is used for flight simulators, computer games, virtual reality and visualization. Restricted by rendering ability of image hardware, the main target of early algorithms is to simplify the terrain description and reduce rendering triangles. Such as continuous levels of detail(CLOD) rendering of height fields[1,2], real-time optimal adaptive meshes(ROAM) algorithm[3], view dependent progressive meshes(VDPM) algorithm[4], and so on. For the purpose of real-time rendering, these algorithms effectively reduce the number of triangles by using LOD technique. But, the disadvantage of these algorithms is to expend large amounts of run-time of CPU.

Along with the developing of GPU and image hardware, triangle rendering number is not the chief bottleneck of modern image system anymore. Many parallel rendering algorithms presented in recent years, such as geometrical mipmap(Geomipmap), chunked LOD and geometrical clipmap algorithm. These algorithms utilize high-speed catches of displaying hardware, increase rendering speed, and save CPU run-time, improve the system performance.

Up to now, for the latest displaying card, GPU CUDA cores has been 2048 and displaying memory has been 4GB too. How to adequately utilize displaying hardware has been an important way for terrain rendering. The method in this document is to compute the error metric, LOD selection and triangulation by GPU, save the CPU run-time. So that CPU can do it’s obligatory work, such as impact detection, game computation, and so on.

Related Work

Error Saturation

An optimal output-sensitive triangulation algorithm is presented in [6] that exploits the strict error monotonicity achieved by a technique known as error saturation. This allows for a simple top-down vertex selection algorithm which does not have to resolve any restricted quadtree dependencies or propagate triangle splits at run-time. This error saturation is performed in the preprocess: Each vertex stores the maximum value of all propagated errors and its own computed error, and propagates this maximum further along the dependency graph. This preprocess can be implemented by a simple traversal over the grid-digital elevation values. Therefore, a fast top-down selection of vertices according to their saturated error metric directly yields an adaptive and conforming triangulation of a restricted quadtree, without the need of enforcing any quadtree constraints, forced splits or resolving dependency relations.

Error Metric

Both simplification of terrain data and LOD technique are based on object space error or screen space error.

In [2] the definition of an efficient view-dependent image-space error metric has been proposed by Lindstrom, which determines removal or inclusion of vertices for a given viewpoint. The basic idea of this error metric is that triangle pairs can be merged if the change in slope at the removed base vertex v projected into screen space is smaller than a given threshold τ. The line segment ε between the removed base vertex v and its linear interpolation is perspectively projected onto the screen space viewing plane as ρ_v. If ρ_v is smaller than the tolerance τ then the vertex v can be removed and the corresponding triangle pairs merged. But the metric defined in [2] is hardly noticeable in practice.

Pajarola proposed an accurate object-space geometric error metric computing method with RQT In [6]. Error values of vertexes in triangle grid T are propagated and maximized along the dependency graph shown in Figure 1. The maximum error ensures that every vertex conform to error saturation and obtain accurate object-space geometric error.

Then, Lindstrom presented an error metric algorithm, which is based on the accurate object-space geometric error of [6] and nested sphere hierarchy of [10], to transform the nested object-space geometric error into nested screen-space error for terrain simplification. This algorithm ensures both object-space geometric errors and screen-space errors are saturated.

Moreover, Today's GPUs are able to sustain speeds of hundreds of millions of triangles per second. So, terrain rendering algorithms mostly utilizes the performance of GPU. [7, 11] partition the terrain into square patches tessellated at different resolutions, [12] directly loads compressed terrain data into graphics memory, and [13] still provides a seamless connecting method between patches. Those algorithms use batch technology of CPU and cache of graphics memory, largely improve rendering efficiency. But [7] only computes static error, [11] largely simplifies error, and [12] even takes the distance to view point for error metric. So, it is still difficulty to take quick, Parallel-based, optimized terrain rendering effect.

GPU-Base Error Metric

On the basis of paper [8], our method computes saturated error of each vertex in terrain grid patch, and compose error map, which is loaded into graphic memory with terrain elevation map. Then we project geometric error to view-dependent screen space error of each vertex, and select vertexes to compose triangle strip by GPU according to the screen space error.

Nested Geometric Error And Screen Error

Nested error means that error increases monotonic. To guarantee this property, we could define the geometric error of vertex is the maximum error of all DAG children. Let

$$\varepsilon_i = \begin{cases} \hat{\varepsilon}_i & \text{if i is a leaf node} \\ \max\{\hat{\varepsilon}_i, \max\limits_{j\in C_i}\{\varepsilon_j\}\} & \text{otherwise} \end{cases}$$

where $\hat{\varepsilon}_i$ is the actual (not necessarily nested) geometric error measured by the object space metric. And $j \in C_i$ means vertex j is one of all DAG descendants of vertex i.

Therefore, it is not sufficient to nest the object space errors alone, but it is necessary to account for this spatial relationship between parent and child vertices. Instead, we resort to a more easily expressed superset of points to project from, defined by a ball $B_i \supseteq P_i$ of radius r_i centered on $\mathbf{p}_i$ (position vector of vertex i):

$$B_i = \{x : \|x - \mathbf{p}_i\| \le r_i\}$$

Where radius r_i of sphere B_i is:

$$r_i = \begin{cases} 0 & \text{if i is a leaf node} \\ \max_{j \in C_i}\{\|\mathbf{p}_i - \mathbf{p}_j\| + r_j\} & \text{otherwise} \end{cases} \qquad (1)$$

Then $B_i \supseteq P_i$ for $j \in C_i$, i.e. the ball hierarchy is nested.

Finally, the maximum projected error was defined as

$$\rho_i = \rho(\varepsilon_i, B_i, \mathbf{e}) = \max_{x \in B_i} \rho(\varepsilon_i, x, \mathbf{e})$$

Because $\varepsilon_i \geq \varepsilon_j$, $B_i \supseteq B_j$, and ρ is monotonic, so we have $\rho_i \geq \rho_j$ for $j \in C_i$. Consequently, if j is active, then so is its parent i, which is what we set out to show.

Geometric Error Metric

Longest edge bisection is shown in Fig.1. Where v is the split vertex, and $e = \{v_l, v_r\}$ is the split edge. The two triangles that share split edge are called the diamond $T = \{t_b, t_t\}$.

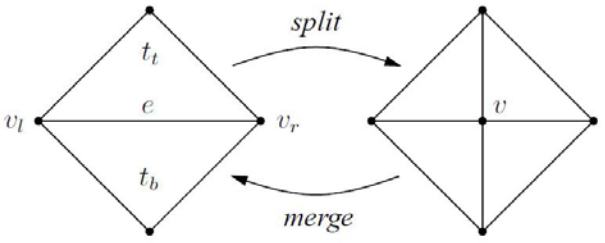

Fig.1. Longest edge bisection

So, we define the vertical error between vertex i and it's triangle t as

$$\hat{\delta}_{i,t} = |z_i - z_t(x_i, y_i)|$$

where z_i is the elevation of vertex i, and $z_t(x_i, y_i)$ is the elevation of triangle t at the point (x_i, y_i) in the domain. So, the incremental error can then be written as

$$\hat{\varepsilon}_i^{inc} = \max_{t \in T_i}\{\hat{\delta}_{i,t}\} = \left| z_i - \frac{z_l + z_r}{2} \right|$$

The maximum error can similarly be written by considering i and all of its descendants:

$$\hat{\varepsilon}_i^{\max} = \max_{t \in T_i}\left\{\hat{\varepsilon}_i^{inc}, \max_{t \in T_i} \max_{j \in D_{i,t}}\{\hat{\delta}_{i,t}\}\right\} \qquad (2)$$

where $D_{i,j}$ is the set of all descendants of *i* reached via recursive bisection of triangle *t*. By the equation (2), the nested geometric error of each vertex in terrain grid can be computed during pre-processing. And the nested screen error also can be projected by the way of bounding sphere.

Screen Error Metric

Given an object space measure of error ε, a view-dependent algorithm projects ε onto the screen, resulting in a screen space error $\rho(\varepsilon)$. For our projecting method, only the Euclidean distance is taken into account, which is between the viewpoint $\mathbf{e}$ and the vertex position $\mathbf{p}$.

$$d = \|\mathbf{e} - \mathbf{p}\|$$

Then the relevant error projection metric of this form can be written as

$$\rho(\varepsilon, \mathbf{p}, \mathbf{e}) = \lambda \frac{\varepsilon}{\|\mathbf{e} - \mathbf{p}\|} = \lambda \frac{\varepsilon}{d} \qquad (3)$$

i.e. the projected error decreases with distance from the viewpoint. λ is a performance parameter, it is

$$\lambda = \frac{w}{2 \tan \varphi / 2}$$

where w is the number of pixels along the field of view φ. Equation (3) is in actuality a projection onto a sphere and not a plane, so a more appropriate choice is $\lambda = w/\varphi$. We then compare ρ against a user-specified screen space error tolerance τ.

In our ball hierarchy refinement procedure, we need to find the maximum projection $\rho(\varepsilon, B, \mathbf{e})$ over a set of points B (Section 3.1). For Equation (3) the maximum projection occurs where $d = \|\mathbf{x} - \mathbf{e}\|$ is minimized. For viewpoints inside B, this term is zero, and we activate the vertex. If $\mathbf{e} \notin B$, then the minimum is $d - r$, and our maximum screen space error becomes

$$\rho(\varepsilon, B, \mathbf{e}) = \max_{\mathbf{x} \in B} \rho(\varepsilon, \mathbf{x}, \mathbf{e}) = \lambda \frac{\varepsilon}{d - r} \tag{4}$$

Comparing ρ against τ and rearranging and squaring some terms (to avoid costly square roots), we obtain

$$\begin{aligned} active(i) &\Leftrightarrow \rho(\varepsilon_i, B_i, \mathbf{e}) > \tau \\ &\Leftrightarrow \lambda \frac{\varepsilon_i}{d_i - r_i} > \tau \\ &\Leftrightarrow (v\varepsilon_i + r_i)^2 > d_i^{\,2} \end{aligned} \tag{5}$$

Where $v = \frac{\lambda}{\tau}$ is constant during each refinement. In a strict mathematical sense, the derivation of equation (5) is valid only if our assumption $\mathbf{e} \in B_i$ holds. However, if we use the convention that $e \in B \Leftrightarrow active(i)$ or $d_i \le r_i \Leftrightarrow active(i)$, then Equation (5) can correctly be used for all viewpoints.

The above expression involves only six additions and five multiplications, and corresponds to parallel computing requires. Therefore, error metric and active vertexes can be computed by GPU.

Parallel-based Error Metric

For massive terrain data, we partition the terrain mesh into equal square tiles (size of 257×257), which elevation data are saved by terms of array. Then we compute the nested geometric error of each vertex of every tile according to equation (2) during pre-processing, and saved by array too. Moreover, in order to project nested screen space error, we compute a nested bounding sphere radius array for all tiles. The process is shown in Fig.2.

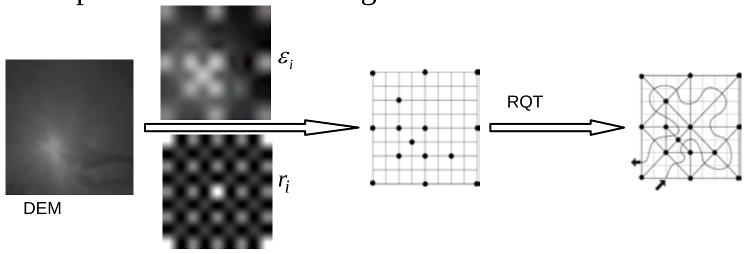

Fig.2. Longest edge bisection

During terrain rendering, the elevation data and error data of required terrain tiles are loaded into graphic memory on the base of view point. Then we can compare projected screen error against tolerance to select vertexes by GPU.

Therefore, we compare one by one to get the LOD active vertexes according to the view point, and construct the triangle strip by means of RQT to render quickly.

Seamless Connection of Tiles

The main challenge for tiled terrain rendering method is to seamlessly stitch block boundaries. There are several ways to fix the cracks between tiles: to add flanges around each block [7], to render zero-area triangles along the tile boundaries [12], and [16] is to stitch adjacent patches by four special strips.

In this paper, we uniform the geometric errors of vertexes, which are on the adjacent edges (E) of two adjacent tiles (A and B). And the consistent error is set to be the maximum one of the two tiles:

$$\varepsilon_v = \max\{\varepsilon_{v,A}, \varepsilon_{v,B}\} \quad v \in E$$

Moreover, for the require of monotonicity, edge error should propagate to its parent too.

Therefore, there are the same geometric error, same bounding sphere radius and same distance to view point, of the same edge vertex of two adjacent tiles. So, there are the same projected screen space errors by means of equation $\rho(\varepsilon, B, \mathbf{e})$. Then, the same vertex of two adjacent tiles could be selected and rendered at the same time, to ensure the connection between tiles is seamless. Moreover, it doesn't require extra run-time work, which only spends some additional computing to get the same edge error during pre-processing.

Test results

We used a 2.6GHz Pentium Dual-Core E5300 PC, with 2G DDR2 of RAM, GeForce 9800GT graphics with 1G of graphics RAM, and SATA 250G disk. For all results, a data set over the Puget Sound area in Washington was used, which is made up of 16384×16384 vertices at 10 meter horizontal and 0.1 meter vertical resolution, and the texture data used pseudo-color.

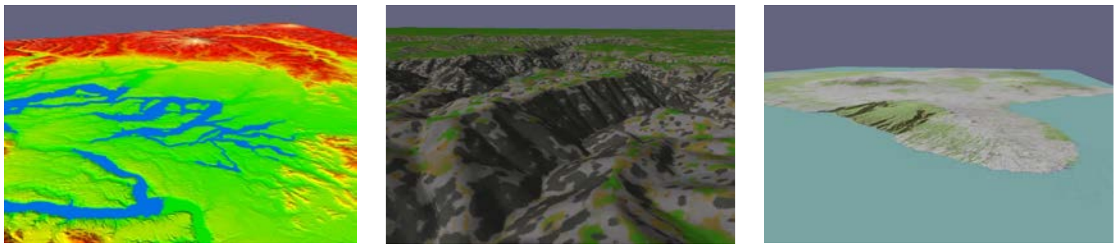
(a) Puget Sound area (b) California Mountains (c)Hawaii island

Fig.3. Terrain rendering by our method

Our application is implemented by VC and OpenGL, and the window size is in all cases 1024×768 pixels with 1 pixel screen space error thresholds. Application image is as Fig.3, and the result of 1000 frames is shown in Fig.4.

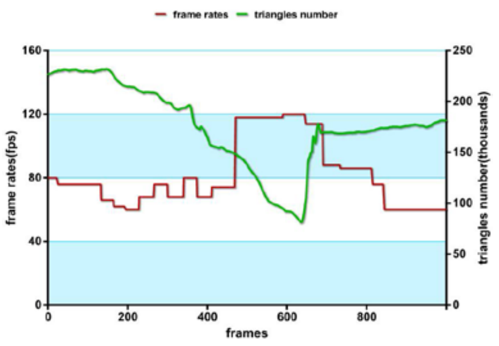

Fig.4. Examination result

The frame rates are between 60 to 120 fps, and the number of triangles is 150 thousands in average. Results from our analyses suggest that our method largely reduce the triangles number, and frame rates reached the real-time requirement of our travels.

Conclusion

In summary, a terrain rendering technique for large, textured terrain is presented here, which uses Parallel-based error metric. By this method, the works of error metric computing is transferred from CPU to GPU without any rendering quantity reduction, which sufficiently utilizes the computing ability of GPU while leaving the CPU free to work on other important tasks. In order to obtain the seamless connection, we also keep the error consistency of two connecting tiles. The result meets our rendering requirement. But, our method increases about 10% sizes of out-of-core files, which restricts the rendering efficiency in a certain extent. For the future, a method to improve loading speed of terrain files is necessary for the purpose of increase farther rendering speed.

References

[1] D. Koller, P. Lindstrom, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner, “Virtual GIS: A real-time 3D geographic information system,” In Proceedings Visualization 95, pp. 94–100. IEEE, Computer Society Press, Los Alamitos, California,1995.

[2] 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, pp. 109–118. ACM SIGGRAPH, 1996.

[3] Duchaineau M, Wolinsky M, Sigeri D E, et al. “Roaming terrain: real-time optimally adapting meshes,” In Proceedings of IEEE Visualization’97 [C]. Phoenix, pp. 81-88, 1997.

[4] Hoppe H. “Smooth view-dependent level-of-detail control and its application to terrain rendering,” In Proceedings of IEEE Visualization’98 [C]. New Caledonia, pp. 135-142, 1998.

[5] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein, “ROAMing terrain: Real-time optimally adapting meshes,” in IEEE Visualization ’97, Roni Yagel and Hans Hagen, Eds., Phoenix, Arizona, pp. 81–88, Nov. 1997, IEEE.

[6] Pajarola, R, “Large scale terrain visualization using the restricted quadtree triangulation,” In Proceedings IEEE Visualization, pp. 19–26, 1998.

[7] Ulrich, T, “Rendering massive terrains using chunked level of detail,” In Super-size-it! Scaling up to Massive VirtualWorlds (ACM SIGGRAPH Tutorial Notes). ACM SIGGRAPH, 2000.

[8] Lindstrom, P., Pascucci, V., “Terrain simplification simplified: A general framework for view-dependent out-of-core visualization,” IEEE Transaction on Visualization and Computer Graphics 8(3), pp. 239–254, 2002.

[9] Schneider, J., Westermann, R., “GPU-friendly high-quality terrain rendering,” Journal of WSCG 14(1-3), pp. 49–56, 2006.

[10] Thomas Gerstner, “Multiresolution visualization and compression of global topographic data,” Geoinformatica, 2002, To appear. Available as SFB 256 Report 29, University of Bonn, 1999.

[11] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R., “BDAM-batched dynamic adaptive meshes for high performance terrain visualization,” In Proceedings EUROGRAPHICS, pp. 505–514, 2003.

[12] Losasso, F., Hoppe, H., “Geometry clipmaps: Terrain rendering using nested regular grids” ACM Transactions on Graphics 23(3), pp.769–776, 2004.

[13] Yotam Livny, Zvi Kogan, Jihad El-Sana, “Seamless Patches for GPU-Based Terrain Rendering,” Proceedings of WSCG, pp. 201-208, 2007.