An Algorithm of 3D Mesh Reconstructing Based on the Rendering Pipeline

Zhengjie Deng[1, a], Shuqian He[1,b], Chun Shi[1,c], Cuihua Ma[1] and Xiaojian Wu[2]

[1]School of information science and technology, Hainan Normal University, Haikou, Hainan, China

[2]Imofang technology co., ltd, Haikou, Hainan, China

[a]hsdengzj@163.com, [b]76005796@qq.com, [c]605515770@qq.com

Keywords: 3D mesh, Mesh reconstructing, Rendering pipeline, Texture

Abstract. During the using of 3D mesh, the different LOD meshes are needed for a same model. This paper presents an algorithm of 3D mesh reconstructing based on the rendering pipeline. This algorithm bases the pipeline, performs the coordinate transform and records the original coordinates of the mesh during the vertex shader. It interpolates the vertex's 3D coordinates of every pixel in the rasterization. During the pixel shader, it extracts the visible pixels through the depth testing, and then outputs the corresponding 3D coordinates. According to the relative position between the pixels, it connects the 3D vertices together to make a triangle mesh for the current view. The result LOD mesh can be obtained by merging these triangle meshes for different views. The experiments show that the algorithm is effective, speedy and convenient for adjusting the LOD.

Introduction

Recently, the applications of virtual reality become more and more, so the rendering of the virtual object's mesh is necessary in the scene. As higher as people require the fidelity of the virtual scene, the more details are needed to be rendered. In a scene with many objects, when the angle or the distance of the view are changed, the same object should be rendered in a different size region, which occupies different amount pixels. In order to enhance the efficiency, it's better use low LOD mesh, when the occupied pixels are less, so the less vertices are operated, the faster it renders. On the other way, it's better use high LOD mesh for enhancing the details in the more occupied pixels. So the different LOD meshes for a same object are needed to be constructed.

In order to obtain different LOD meshes of a model, users probably base on a mesh with normal granularity, gain other LOD meshes with other algorithms. Some simplify methods can easily create the meshes with lower LOD. [1] presented a greedy approach to simplify those regions which are enriched with number of triangles. [2] made a survey on the most notable available algorithm, which included an overview of iterative edge contraction algorithms. Some subdivide methods can easily create the meshes with higher LOD. In [3], a new stationary subdivision scheme was presented which performs slower topological refinement than the usual dyadic split operation. [4,5] have implemented the subdivision through spline surfaces and piecewise smooth surfaces. But when users want not only the lower ones, but also the higher ones, it is necessary to be familiar with both kinds of methods, and exchange between them frequently.

As the computing of GPU becomes stronger, more and more computations, especially those parallel-able computations, apply GPU for gaining more performance. [6] demonstrated a mesh decimation method, adopted a vertex-clustering method to GPU by taking advantage of the geometry shader stage. It presented a novel general-purpose data structure designed for streaming architectures.

In this paper, we presents a 3D mesh reconstructing algorithm based on the rendering pipeline. Focus on adjusting LOD, it does an LOD decision method through the view distance, which can carry out visually. During the computation, it applies the rendering pipeline of GPU, create the target mesh quickly with the GPU's parallel computing. The main contributions are as following.

(1) Record the coordinates of the mesh's vertices through an A32B32G32R32 format texture;

(2) According to the angle of the view, generate the 3D vertices and arrange them in an order like texture's pixels, which make it easy to organize them into a mesh.

The rest parts of the paper are arranged as: Section 2 reviews the traditional methods to create different LOD meshes; Section 3 illustrates the main idea of the algorithm in this paper; Section 4 introduces the normal progress of the rendering pipeline; Section 5 presents how to reconstruct the mesh based the rendering pipeline in detail, and how to implement the algorithm, including the vertex shader and the pixel shader; some experiments using this algorithm are shown in the Section 6. At last, Section 7 makes a conclusion and discusses the future about it.

Traditional LOD Mesh Reconstructions

When the LOD requirement is lower than the original one, it means that the new LOD mesh is a simpler version of the original one. In order to make the new one is as similar as the original one, many simplifications apply the Eq. 1 to obtain a new vertex during simplifying.

$$L(u,v) = \min Dist(S,S(L(u,v))), \tag{1}$$

while S is the original mesh; u and v are the parameterize coordinates on S; $L(u,v)$ is a new vertex about the area around (u,v), which is used to preplace a group of vertices, edges or patches on S; $S(L(u,v))$ is the interval mesh that is obtained through using $L(u,v)$, and the distance function *Dist* is used to calculate the difference between S and $S(L(u,v))$. Usually, the function bases on local area. Different simplifications provide different distance functions.

When the LOD requirements are higher than the original one, it seems that the mesh need subdividing. In order to let the refined result be similar to the original one, many subdivisions apply the Eq. 2 to generate the new vertex.

$$H(u,v) = Interpolate(S,u,v), \tag{2}$$

while the subdivide function *Interpolate* would do the interpolation based on the area around (u,v), to generate the new vertex $H(u,v)$.

The computation in the Eq. 1 and Eq. 2, often directly use the local vertex data of the original mesh, to produce the new vertex for the corresponding area. The final result of the two kinds of methods is a new mesh T, and make T and S be similar, while T accepts the special LOD requirement.

The Main Idea of the Algorithm

If there has been a mesh T accepting the requirement, the next work is only to sample T through (u,v). Fortunately, when the model is shown to the user, GPU has implemented the function that created a view mesh T_i, which viewed the T through the current view angle. So, T can be generated by merging the T_is together, which are created quickly by GPU's rendering. In this way, the main idea of our algorithm follow through the Eq. 3.

$$\begin{aligned} &T_i = View(S,i), \quad (i=1,\dots,n), \\ &T = Merge(\{T_i\}), \quad (i=1,\dots,n), \end{aligned} \tag{3}$$

while the function *View* is to create the view mesh T_i, which view the original mesh S through the ith view angle; n is the amount of the view angle according to the real condition; the function *Merge* generates the new mesh T through merging all view meshes.

Our algorithm seems to parameterize a mesh, and then sample it based on a special interval. But be different to the traditional parameter method, it directly parameterizes through the projection during rendering, to make the result be same to the original mesh from the view angle. The advantages of our algorithm are as following:

(1) Simply operating: only adjust the view port to specify the LOD;

(2) Speedy: based on the rendering pipeline, generate the target mesh in the interactive time;

(3) High similarity: the view of new result is same to the magnifying result of the original model.

Rendering Pipeline

As the basic procedure of the rendering pipeline of the computer display, let the world transform matrix of the object be *W*, view transform matrix be *V*, project transform matrix *P*, and the matrix magnifying to the screen be *E*, organize the vertex coordinates of *S* into matrix, named by *S* too, so let the new mesh be *SE* in the screen space transformed from the original *S*, through Eq. 4.

$$SE = S*W*V*P, \tag{4}$$

Anyway, the LOD of *SE* is still same to *S* at this time. In order to generate the mesh with a special LOD, the user should specify a LOD for the special application. In the real life, the farer the user leaves the object, the lower LOD is required, vice versa. Directly, our method supports users to decide the LOD through adjusting the distance away the object. In the Fig. 1, user can adjust the distance away the car, and then decide how many pixels would be occupied by the car. The less of pixels as Fig. 1 (a), the lower LOD; the more of pixels as Fig. 1 (c), the higher LOD.

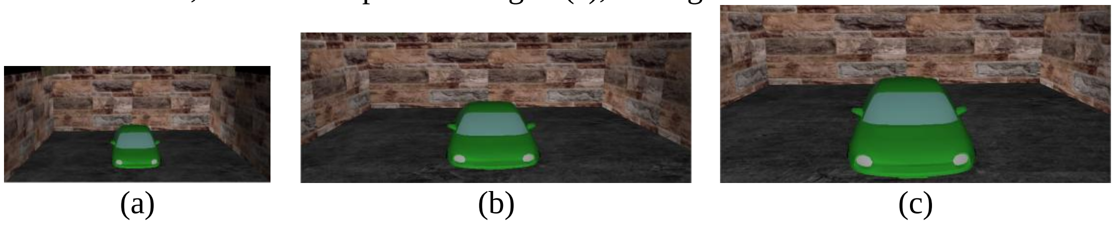
(a) (b) (c)

Figure 1: Decide the LOD through adjusting the view distance.

Next, according to the size of the view port and the distance, the LOD sampling is applied. The rasterization of GPU is applied through the Eq. 5 to obtain the sample result *SR*.

$$SR = Rasterize\ (SE), \tag{5}$$

while the function *Rasterize* means to rasterize *SE* based on the current view port, and obtain the pixel set *SR*.

During the implement of the function Rasterize, it likes the function Interpolate in Eq. 2, to compute the new vertex's position through vertex interpolation. This makes sure the similarity of *SR* and *SE*. On the other way, the function Rasterize is different to the function Interpolate, to generate the different LOD vertex set, which is on the surface that is made of the visual parts of the object in the current view. The vertices sheltered from others would be filtered out during the depth testing.

Mesh Reconstructing Based on the Rendering Pipeline

Mesh Reconstructing. GPU's rendering pipeline can quickly finished the computations of Eq. 4 and Eq. 5, which generates *SR*. But the coordinates of the target mesh should be in the coordinate space of the original mesh *S*. It means that *SR* should be restored as a mesh *S'* in the original space, which can be computed by Eq. 6.

$$S' = SR*P^{-1}*V^{-1}*W^{-1}. \tag{6}$$

If the restoring can be moved before the function *Rasterize*, the original *S* is passed into *Rasterize*. It would save many performance. Unfortunately, *Rasterize* could not compute on the vertices in the object space. A good news for us is that GPU supports to attach textures to a mesh, with which every vertex can attach one or more texture pixel colors. During the vertex transforming, the pixel colors do not be transformed, but *Rasterize* can interpolate the texture color value according to the coordinates. And almost display cards support the texture format A32B32G32R32 now. The pixel of this kind of texture can contain the normal 3D coordinates of one vertex. So, in order to skip the restoring, this algorithm attaches an A32B32G32R32 texture to *SE*, and the pixel's values are the coordinates of the corresponding vertex of *S*. In this way, the computation about transforming and restoring can be replaced by Eq. 7.

$$(SR, SN) = \text{Rasterize}((SE, SE.T)),\quad SE.T = S, \tag{7}$$

while (*SE*, *SE.T*) is an *SE* with the texture *SE.T*, the values of *SE.T* come from the coordinates of *S*. After rasterization, *SN* is the vertex set in the object space. *SR* is the result sampling from *SE*, and *SN* is the corresponding result sampling from *S*.

Viewing from the current angle, *SN* is a vertex set, which are arranged likes pixels in a texture. Connecting the neighbor vertices can construct a triangle mesh. If constructing with a background plane, which is a plane farer than the object, or the object's bounding box, a mesh T_i for the current view angle can be generated. For different models, users can construct for different angles, and then obtain all angle's T_i. Usually, it can be constructed according to the normal of six faces of the bounding box. And then, merging the meshes together to make the new LOD mesh for the object. For those objects not convex, it may construct more T_is.

Implementing the Algorithm. The application illustrating the results in this paper is developed with Microsoft's Direct3D library. Eq. 4 and Eq. 7 are implemented in the vertex shader. The detail procedure is the function VertDepth.

```
void VertDepth(float4 S:POSITION, out float4 SE:POSITION, out float4 SE_T:TEXCOORD1)
{   SE = mul(S, W);   SE = mul(SE, V);   SE = mul(SE, P);   SE_T = S; }
```

while *SE_T* that is *SE.T* in Eq. 7, is the texture pixel whose value is equal to the original coordinates of the corresponding vertex. The raterization and sampling in Eq. 7 are carried on through GPU's default rasterization function and the pixel shader. The default rasterization is not dicussed here. The procedure of the pixel shader is the function PixDepth.

```
void PixDepth(float4 SE_T : TEXCOORD1, out float4 SN : COLOR)
{   SN = SE_T; }
```

It's a note that the semantic of the output *SE_T* in VerDepth is TEXCOORD1, the one in PixDepth should be same. It's the rule to pass the right value. The PixDepth only outputs *SN*, but not *SR*, because a pixel shader only outputs one color. And the method's target is to generate the coordinates for constructing the new mesh, not for displaying. If user wants to display the object, it can be rendered again in a normal way.

Experiments

The application has been used to construct some meshes for objects. Fig. 2 has shown the results for a sphere object. (a) is the original model; (b) is the lower LOD T_i, which only shows the surface constructed by the vertices of *SN*, without the background or the bounding box for clearly (the same for the later); (c) is the higher LOD T_i; (d) is the another view of (c) from another angle; (e) is the result merging two T_is.

The result of the sphere shows that for a mesh, our method can efficiently create its different LOD meshes, and keep similar to the original one. Note that the boundary of the obtained mesh is not regular, because the object covers itself during viewing.

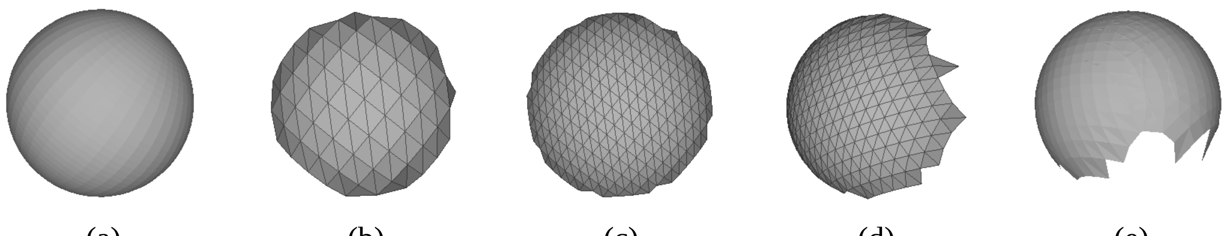

Figure 2: Reconstructing the sphere's mesh.

The method has been used on more complex model, and also obtain a good result as Fig. 3. (a) and (c) are the two views of the original model. (b) and (d) are the meshes constructing from the car's front and back. In order to see them clearly, they are shown with rotating an angle. The result shows that it's similar the original one, and the vertex density is controllable.

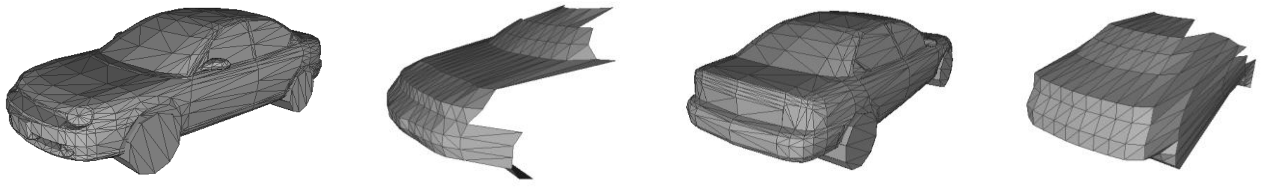
(a) (b) (c) (d)

Figure 3: Reconstructing the car's mesh.

For the performance, the VertDepth and the PixDepth described above show that the algorithm is simple. The VertDepth only contains the ordinary world transform, view transform and project transform. The PixDepth only transfers the coordinate data. When the rendering finishes, the vertex set is generated for the corresponding view angle. Because the vertices in the set are arranged regularly like image's pixels, the mesh can be constructed in linear time.

Conclusions

This paper present a 3D mesh reconstructing algorithm based on the rendering pipeline. It uses a novel vertex shader and pixel shader, makes the rendering result be a vertex set corresponding the view angle. The vertex set is arranged as an image, easily to construct a view mesh. With the view meshes from some different angles, a new mesh can be generated by merging them. The LOD of the new mesh is decided by the view distance, so the LOD can be easily specified. The experiments show that the algorithm is efficient and fast. Anyway, there are some shortages in it now. For example, the side faces of the view mesh sometimes include some long and thin triangles, which may not put out the original model. Our future work would focus on them.

Acknowledgements

This work was financially supported by the National Natural Science Foundation (61502127, 61362016), Hainan Natural Science Foundation (20156225), Educational reform project of Hainan Provincial Department of Education (HNJG2014-33) and Scientific Research Projects in Universities of Hainan Province (Hnky2015-24).

References

[1] S. Hussain, H. Grahn and J. A. Persson: Feature-preserving Mesh Simplification: A Vertex Cover Approach. International Conference on Computer Graphics and Visualization 2008 (CGV 2008), pp270-275

[2] M. Garland: Multiresolution Modeling: Survey & Future Opportunities. EUROGRAPHICS' 99, State of The Art Reports

[3] L. Kobbelt: √3-subdivision. Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co. (2000) p.103-112

[4] E. Catmull, J. Clark: Recursively generated B-spline surfaces on arbitrary topological meshes, CAD 10 (1978), p.350–355

[5]H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, W. Stuetzle: Piecewise smooth surface reconstruction, SIGGRAPH 1994 Proceedings(1994), p. 295–302

[6] C. DeCoro, N. Tatarchuk: Real-time Mesh Simplification Using the GPU. Symposium on Interactive 3D Graphics (I3D) (2007), p. 6