SHRINKING LEVELS OF GENERIC PROGRESSIVE MESHES WITH TRIANGLE BUDGET OR ERROR CONTROL

Shu-Kai Yang, Chin-Chen Chang, Ming-Fen Lin, Ding-Zhou Duan

Computer & Communications Research Laboratories

Industrial Technology Research Institute Hsinchu, Taiwan, R.O.C.

Email: *sagitta@itri.org.tw*, *chinchen@itri.org.tw*, *mingfen@itri.org.tw*, *dwin@itri.org.tw*

ABSTRACT

In modern virtual reality applications in electrical commerce or virtual environments, triangular meshes are often considered to be multi-resolutional for transmitting and displaying complex models through Internet or rendering large-scaled scenes in real-time. There are already some mesh simplification algorithms developed to generate progressive meshes that convert a single-resolution triangular mesh into a base mesh with a sequence of refinement operations. Sometimes the length of sequence might be too long and the geometric modification of a single refinement step might be too slight. In this paper, we present a generic progressive mesh representation so that we can convert progressive meshes generated by different algorithms into the same representation. We also present a series of post-processing rules to shrink the refinement sequences of generic progressive meshes with the triangle budget or error controlled for each level. We call the procedure level shrinking. It improves the efficiency of level-of-detail modulation of progressive meshes if they are supposed to be used in real-time rendering systems.

1.INTRODUCTION

The obvious integral trend of virtual reality is to build virtual environments that may be integrated with networking technology or several kinds of input device. Scientists may perform simulations in virtual environments rather than in the real world to reduce the cost and material wastage. Home users may simulate actions in virtual environments for consumption or entertainment. All these functions described above require the ability of systems in rendering large-scaled scenes in real-time. Most virtual environment systems are integrated with visibility culling techniques to avoid the unnecessary computations and rendering cost of graphics hardware. During the screen displaying for a user in an environment, systems may only render visible portions of the scene. Sometimes the visible objects are composed of complex models, so the cost of rendering these objects is still too expensive and the frame rate is still too slow. Level-of-detail technology is exploited in virtual environment systems to solve such situations. It is to use multi-resolutional models such as progressive meshes instead of single-resolutional complex models in the scene, to render visible objects near the user's viewing point in finer resolutions, and to render those far away from the user's viewing point in coarser resolutions.

After applying a simplification process to a triangle mesh, we get a coarse mesh called *base mesh* in which the features of the original mesh should be highly preserved. We also get a sequence of refinement operations by recording the modification of every simplification step. The sequence can be used to convert the base mesh to the original mesh. A base mesh together with a refinement sequence is called a *progressive mesh*. If a progressive mesh is composed of a base mesh and *n* refinement operations, we say that the mesh has *n+1 resolution levels*. Converting models into progressive meshes is a preprocessing procedure, and virtual environment systems decide the rendering resolutions of models in run-time. For this usage, the data structure of progressive meshes should be efficient for resolution modulation, so we have to be able to control the amount of resolution level of progressive meshes. Besides, systems may decide resolutions of models according to their distances to the user's location and their errors to the original models. System may also have to control triangle amounts of models because of limitation of the rendering ability of hardware. So we need to possess the triangle amount and error of each level of every progressive mesh. Sometimes the production of mesh simplification algorithms does not match our requirements. Maybe the length of refinement sequence of a progressive mesh is too long and it takes to much time to modulate the level of details of the mesh. In this paper, we present a generic progressive mesh representation so we can convert progressive meshes generated by different algorithms into the same representation. For the generic progressive mesh representation, we also present a procedure to reduce the amount of levels of a mesh with the triangle amount or error of each level possessed. By exploiting the procedure, we can improve the efficiency of resolution modulation of progressive meshes generated by any algorithm to fit our uses in virtual environment systems.

2. RELATED WORKS

2.1. Vertex Clustering

As shown in Figure 1, the method uniformly divides the space occupied by a triangle mesh into cells, selects a representative vertex with the highest visual importance for each cell, and merges all vertices to these representative vertices [4]. The method may eliminate lots of vertices in single step, but the preservation of model features is not good because it does not keep more vertices at these characteristic portions.

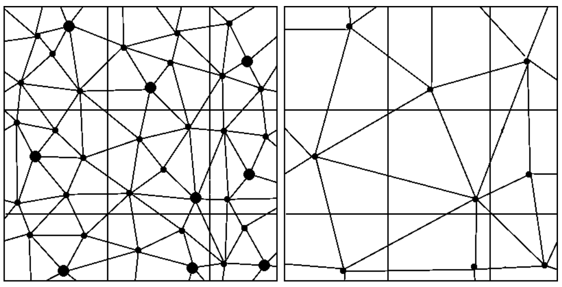

Figure 1: Vertex clustering.

2.2. Edge Collapsing

For each edge of a model, the method evaluates the cost of merging its two end vertices and sorts all edges according to their cost values. Then it selects the edge with the lowest cost and merges its two vertices repeatedly [10]. The operation shown in Figure 2 is called an *edge collapsing*. Its inverse operation is called *vertex split*. A simplified mesh together with a sequence of vertex split operations is the representation of progressive meshes generated by edge-collapsing algorithms [8].

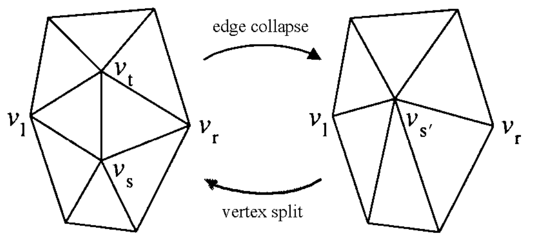

Figure 2: Edge collapsing and vertex split.

Edge-collapsing algorithms make good preservation for model features but one edge collapsing removes two triangles at most. If a mesh is composed of m triangles and is simplified into a coarse mesh of m_0 triangles, the generated progressive mesh will have $n=(m - m_0)/2$ vertex splits at least because one vertex split increases two triangles at most.

2.3. Error Estimation

The Euclidean distance between a point x and a set Y is defined by

$$d(x,Y) = \inf_{y \in Y} d(x,y), \qquad (2.3.1)$$

where d(.,.) is the Euclidean distance between two points in R^3. We can define the distance $d_E(X,Y)$ from a set X to a set Y by

$$d_E(X,Y) = \sup_{x \in X} d(x,Y). \qquad (2.3.2)$$

The Hausdroff distance $d_H(X,Y)$ in R^3 is defined by

$$d_H(X, Y) = \max(\, d_E(X, Y), d_E(Y, X)\,). \qquad (2.3.3)$$

We may use the Hausdroff distance between a simplified mesh and the original mesh as the error of the simplified mesh [11]. Some people also use the maximum distance of vertex shifting as the error caused of mesh simplification. There are already some algorithms developed to estimate the difference between a simplified mesh and the original mesh.

2.4. Vertex Decimation

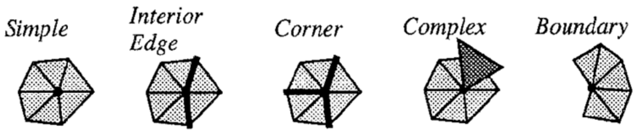

Figure 3: Vertex classification.

The method classifies all vertices into five categories listed in Figure 3. Then it removes simple vertices that are nearly co-planar with their adjacent vertices and re-triangulates the holes caused of these vertex removals. In other words, the method replaces some sub-surfaces of the original mesh by sub-surfaces composed of fewer triangles. Some algorithms use edge collapsing to close the holes caused of vertex removals instead of re-triangulating them. A progressive mesh generated by vertex decimation algorithms could be a base mesh together with a sequence of inverse sub-surface replacements.

3. GENERIC PROGRESSIVE MESH REPRESENTATION

The notion of progressive meshes described in this paper can be written as

"Triangular meshes whose resolution can be modulated by modifying partial geometry."

For a progressive mesh M, M^l denotes the mesh M in level l. Let M^0 denote the base mesh. R_l denotes the refinement that makes M^{l-1} become M^l. C_l denotes the inverse operation of R_l. Using these notations, a progressive mesh M can be treated as a finite state machine shown in Figure 4. A progressive mesh M with n refinements has $n+1$ states and can change its state via R_l and C_l where $l = 0, \ldots, n$.

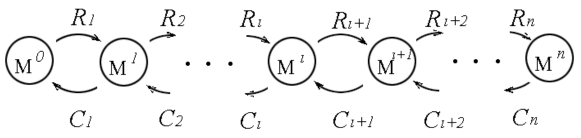

Figure 4: Mesh states.

Let t_i^l denote the triangle of index i in the state of level l. It is trivial that that if R_{l+1} does not have any operation on t_i^l then $t_i^{l+1} = t_i^l$. We define the three following commands:

delete(t) : to remove triangle t from the current mesh,

add(t): to add a new triangle t to the current mesh,

replace(t, u, v) : to replace the corner vertex u of triangle t by vertex v.

We can categorize progressive meshes generated by all kinds of mesh simplification algorithms into two categories. One is *vertex-split progressive meshes* which may be generated by some edge-collapsing or vertex-clustering algorithms. The refinement operation for vertex-split progressive meshes is splitting some vertices to open some holes on the mesh and fitting some new triangles into these holes. Such an operation can be achieved by using *replace* and *add* commands. R_l in such progressive meshes can be written as "*replace*($t_i^{l\text{-}1}$, *u, v)* for some indices i included in $M^{l\text{-}1}$ and *add*(t_j^l) for some indices j not included in $M^{l\text{-}1}$". The other category of progressive meshes is *subsurface-replacing progressive meshes* which may be generated by vertex decimation algorithms. The refinement operation for subsurface-replacing progressive meshes is removing some sub-surfaces and fitting finer sub-surfaces into the holes caused of these removals. Such an operation can be achieved by using *delete* and *add* commands. R_l in such progressive meshes can be written as "*delete*($t_i^{l\text{-}1}$) for some indices i included in $M^{l\text{-}1}$ and *add*(t_j^l) for some indices j not included in $M^{l\text{-}1}$". Considering these two categories of progressive meshes and hybrid cases, we describe our generic progressive mesh representation as

"A base mesh together with a sequence of refinement operations that are composed of delete, replace, and add commands."

We can say that every refinement R_l is a set of commands such as *delete*($t_i^{l\text{-}1}$), *replace*($t_i^{l\text{-}1}$, *u, v*), and *add*(t_j^l) for some indices i included in $M^{l\text{-}1}$ and for some indices j not included in $M^{l\text{-}1}$.

4. LEVEL SHRINKING

In the previous section we have discussed that a progressive mesh can be treated as a finite state machine. To shrink the refinement sequence of a progressive mesh can be thought as to remove some states of the finite state machine. As shown in Figure 5, we delete state M^l and derive R'_{l+1} from R_l and R_{l+1} for the progressive mesh M. We can shrink the refinement sequence of a progressive mesh by performing such state deletions repeatedly.

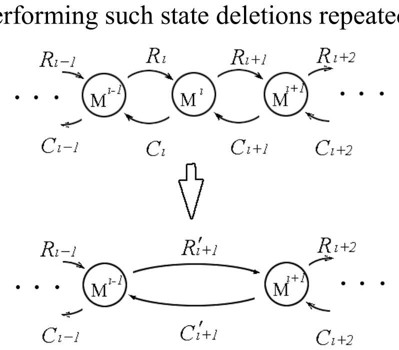

Figure 5: Removing state M^l of progressive mesh M.

4.1. Cases and Rules

To delete state M^l, we merge R_l and R_{l+1} and create R'_{l+1}. Since the input of R'_{l+1} is M^{l-} and the output is M^{l+1}, any triangle t_i touched by R'_{l+1} does not need to pass through the state of t_i^l and can be converted to t_i^{l+1} directly. Some commands in R_l and R_{l+1} processing the same triangles may be eliminated or converted while creating R'_{l+1}. Table 1 lists the cases and their corresponding rules of re-writing commands in R_l and R_{l+1} processing the same triangle t_i. In case 1 through case 6, if t_i is only touched by one of R_l and R_{l+1}, we have to duplicate these commands in R'_{l+1}. In case 7, if we add t_i in R_l and may apply vertex replacements in R_{l+1}, we only have to add t_i which is already after vertex replacements directly. In case 8, if we apply vertex replacements that do not replace the same vertices both in R_l and R_{l+1}, we can not eliminate any one of these commands. But in case 9, if some vertex replacements replace the same vertices, we only have to put commands that replace vertices of $t_i^{l\text{-}1}$ by vertices of t_i^{l+1} directly in R'_{l+1}. In case 10 and 11, if t_i is deleted in R_{l+1}, all commands processing t_i in R_l can be canceled in R'_{l+1}.

t_i	**Commands**		
Case	**R_l**	**R_{l+1}**	**R'_{l+1}**
1	*add*(t_i^l)		*add*(t_i^{l+1})
2		*add*(t_i^{l+1})	*add*(t_i^{l+1})
3	*replace*($t_i^{l\text{-}1}$, *u, v*)		*replace*($t_i^{l\text{-}1}$, *u, v*)
4		*replace*(t_i^l, *u, v*)	*replace*($t_i^{l\text{-}1}$, *u, v*)
5	*delete*($t_i^{l\text{-}1}$)		*delete*($t_i^{l\text{-}1}$)
6		*delete*(t_i^l)	*delete*($t_i^{l\text{-}1}$)
7	*add*(t_i^l)	*replace*(t_i^l, *u, v*)	*add*(t_i^{l+1})
8	*replace*($t_i^{l\text{-}1}$, *u, v*)	*replace*(t_i^l, *x, y*)	*replace*($t_i^{l\text{-}1}$, *u, v*) *replace*($t_i^{l\text{-}1}$, *x, y*)
9	*replace*($t_i^{l\text{-}1}$, *u, v*)	*replace*(t_i^l, *v, w*)	*replace*($t_i^{l\text{-}1}$, *u, w*)
10	*replace*($t_i^{l\text{-}1}$, *u, v*)	*delete*(t_i^l)	*delete*($t_i^{l\text{-}1}$)
11	*add*(t_i^l)	*delete*(t_i^l)	

Table 1: Cases and rules for creating R'_{l+1}.

4.2. Benefits

The cost of refining a mesh is related to the amount of commands required, not related to the amount of resolution levels. If we can not reduce the amount of commands, we do not get any benefits in a level shrinking procedure. Considering the cases listed in Table 1, in case 1 though case 6 and case 8, any command required by R_l or R_{l+1} is also required by R'_{l+1}, so we can not get any benefits. In case 7, 9 and 10, we merge pairs of commands in R_l and R_{l+1} to single commands in R'_{l+1}, so the cost is reduced to a half. In case 11, we do not leave any command in R'_{l+1}, so the cost is avoided completely. Case 7 though 11 occur while the modifying areas of R_{l+1} overlaps those of R_l and the level shrinking procedure earns benefits in processing these cases.

4.3. Triangle Budget Schemes

For a progressive mesh M with n levels, let m_l denote the triangle amount of M^l and Δm_l denotes the triangle amount increased by R_l. Note that Δm_0 be zero although there is no

R_0. We know that m_0 is the triangle amount of the base mesh and

$$m_l = m_0 + \sum_{k=1}^{l} \Delta m_k \cdot \qquad (4.3.1)$$

Scheme 1: Select a pair of (R_l, R_{l+1}) with the minimum (Δm_l + Δm_{l+1}) and delete state M^l repeatedly until the number of levels is reduced to a user-specified threshold. Then Δm of each level l will converge to a constant in the resulting progressive mesh. In other words, triangle amount m_l will increase linearly in each level l while refining the resulting progressive mesh.

Scheme 2: For a given increasing rate μ of triangle amount, we can derive the triangle budget B_l for each new resolution level l of the resulting progressive mesh by

$$B_l = m_0(1+\mu)^l \qquad (4.3.2)$$

repeatedly until $B_l \geq m_n$ holds, where n is the number of levels of the original progressive mesh and B_0=m_0. For each B_l, find all mesh state M^k satisfying $B_{l-1} < m_k \leq B_l$ and preserve the mesh state with the maximum m_k among these states and delete other states. After the procedure, in each level of the resulting progressive mesh, the triangle amount will be increased with a fixed rate μ while refining the mesh.

4.4. Error Control Schemes

For a progressive mesh M with n levels, let e_l denote the error of M^l. We know that e_0 is the maximum error of M and e_n is zero.

Scheme 1: If we select a pair of (R_l, R_{l+1}) with minimum | $e_l - e_{l+1}$ | and delete state M^l repeatedly until the number of levels is reduced to a user-specified threshold, then the states in which the error of M is reduced rapidly are left. The resolution levels of the resulting progressive mesh will fit in some states of the original state sequence that are most visually different from each other.

Scheme 2: For a given reducing rate μ of error, we can derive the error threshold E_l for each new resolution level l of the resulting progressive mesh by

$$E_l = e_0(1-\mu)^l \qquad (4.4.1)$$

repeatedly until $E_l < e_{n-1}$ holds. We give every mesh state a *preserved* flag which is initialized to be *false*. First, mark the *preserved* flags of M^0 *and* M^n *true*. For each E_l, find the mesh state M_k with the largest e_k satisfying $E_l \leq e_k$, and then mark the *preserved* flag of M^k *true*. Find all mesh state M^k satisfying $E_l < e_k < E_{l-1}$, and delete every M^k among these states whose *preserved* flag is *false*. After the procedure, in each level of the resulting progressive mesh, the error will be reduced by a fixed rate μ while refining the mesh.

4. RESULTS

We have tested the improvement of efficiency of refining progressive meshes after level shrinking for some models and the results are listed in Table 2. Our experimental platform is a PC with an AMD® K6-II 400CPU running Microsoft® Windows 2000 Professional OS. As shown in Table 2, the level shrinking procedure reduces the refining time for each model. How much time reduction we can get depends on the distribution of processing areas of commands in the refinement sequence and the distribution may depend on the mesh simplification algorithms and the curvature properties of the geometry of original models.

Model name	Triangles		levels	Refining time (sec)		
	Max	Base		Unshrinked	1000 levels	100 levels
Beethoven	5030	500	1655	0.000885	0.000828 (-6.44%)	0.000657 (-25.76%)
Cow	5804	498	1602	0.001040	0.000992 (-4.62%)	0.000875 (-15.87%)
Spider	9286	500	1748	0.001865	0.001660 (-10.99%)	0.001340 (-28.15%)
Dog	33885	497	9023	0.005108	0.003922 (-23.22%)	0.003685 (-27.86%)
Bunny	69451	495	8677	0.021500	0.017975 (-16.40%)	0.016250 (-24.42%)

Table 2: Experimental results.

6. CONCLUSION

We present a generic progressive mesh representation and a series of rules of shrinking the levels of generic progressive meshes. We also present some schemes of shrinking levels with triangle budget or error control. The technique can be used to improve the efficiency of resolution modulation of progressive meshes and make them more useful in virtual environment systems.

REFERENCE

[1] "ISO/IEC 14496-2 MPEG-4 Visual Working Draft Version 2 Rev. 5.0", SC29/WG11 document number W2473, 16 Oct. 1998.

[2] Algorri, M. M. and Schmitt F. "Mesh Simplification", EUROGRAPHICS'96, Vol 15, 3(C) :77-86, 1996.

[3] Cohen, J., Olano, M., Manocha, D. "Appearance-Preserving Simplification", Computer Graphics (SIGGRAPH'98 proceedings), 115-122, 1998.

[4] DeHaemer, M. J. and Jr., Zyda, M. J. "Simplification of Objects Rendered by Polygonal Approximations", Computer Graphics, Vol.15, 2:175-184, 1991.

[5] Garland, M. and Heckbert, P.S. "Surface Simplification using Quadric Error Metrics", Computer Graphics (SIGGRAPH'97 proceedings) , 209-224, 1997.

[6] Garland, M. and Heckbert, P.S. "Simplifying Surfaces with Color and Texture using Quadric Error Metrics", IEEE Visualization'98 proceedings, 263-269, 1998.

[7] Gueziec, A., Silva, C., Taubin, G. "A Framework for Streaming Geometry in VRML", IEEE Computer Graphics and Applications, special issue on VRML, March-April 1999.

[8] Hoppe, H. "Progressive Meshes", Computer Graphics (SIGGRAPH'96 proceedings), 99-108, 1996.

[9] Hoppe, H. "View-Dependent Refinement of Progressive Meshes", Computer Graphics (SIGGRAPH'97 proceedings), 189-198, 1997.

[10] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. "Mesh Optimization", Computer Graphics (SIGGRAPH'93 proceedings), 19-26, 1993.

[11] Klein, R., Liebich, G., and Strasser, W. "Mesh Reduction with Error Control", Visualization '96. Proceedings, 311–318, 1996.

[12] Low, K. L. and Tan T. S. "Model Simplification Using Vertex-Clustering", Symposium on Interactive 3D Graphics (1997), 75-81, 1997.

[13] Popoviċ, J. and Hoppe, H. "Progressive Simplicial Complexes", Computer Graphics (SIGGRAPH'97 proceedings), 1217-224, 1997.

[14] Rossignac, J. and Borrel, P. "Multi-resolution 3-D Approximations for Rendering Complex Scenes", In Falcidieno, B. and Kunii, T.L., editors, Modeling in Computer Graphics (1993), Springer-Verlag, Berlin, 455-465, 1993.

[15] Ronfard, R. and Rossignac J. R. "Full-range Approximation of Triangulated Polyhedra" , EUROGRAPHICS'96, Vol. 15, 3:C:67-76, 1996.

[16] Schroeder, W. J. "A Topology Modifying Progressive Decimation Algorithm", IEEE Visualization'97 proceedings, 205-212, 1997.

[17] Schroeder, W. J., J. A. Zarge. And Lorensen, W. E. "Decimation of Triangle Meshes", Computer Graphics (1992), 26(2):65-69, 1992.

[18] Taubin G., Gueziec A., Horn W., Lazarus F. "Progressive Forest Split Compression", Computer Graphics (SIGGRAPH'98 proceedings), 123-132, 1998.

[19] Turk, G. "Re-tiling Polygonal Meshes", Computer Graphics (SIGGRAPH'92 proceedings), 26(2):55-64, 1992.

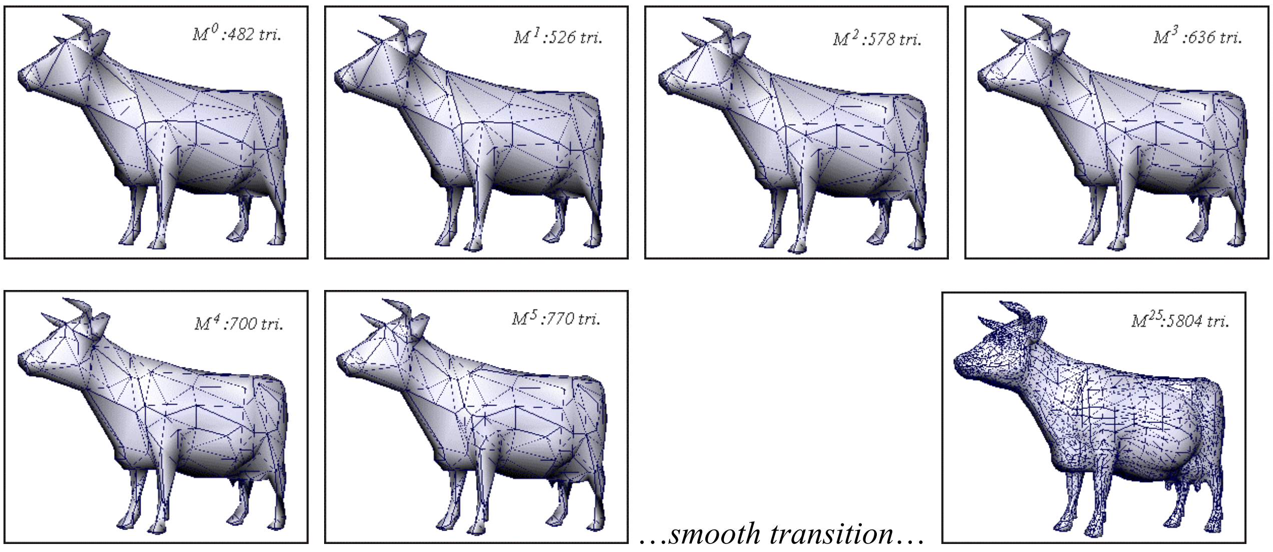

Figure 6: A level-shrunk progressive mesh of a cow model.

(triangle amount increases by 10% for each level)