Scale-Space 3D TexMesh Simplification

Irene Cheng and Pierre Boulanger
Department of Computing Science, Univ. of Alberta, Edmonton, CANADA,{lin,pierreb}@cs.ualberta.ca

Abstract

Efficient on-line 3D visualization is essential for a variety of applications including not only games and e-commerce, but also heritage and medicine. For efficient online visualization, it is necessary to quickly adapt 3D models (both mesh and texture) to the available computational or network resources. In this paper we propose using 3D model simplification based on a scale-space analysis of the surface curvature variations combined with an associated scale-space analysis of the surface texture to reduce the size of texture files, and facilitate distributed transmission. The premise of the proposed simplification is that: minor variations in texture can be ignored in relatively smooth regions of a 3D surface, without significantly affecting human perception. Statistics of feature points and their associated texture fragments are gathered during preprocessing. On-line transmission and rendering for the next higher resolution scale is based on the statistics, which can be retrieved in constant time. Quality of service (QoS) can be provided based on the time limit, number of vertices, or faces requested by the viewer. Experimental results showing the simplified models demonstrate the feasibility of our approach.

1. Introduction

Ever since the introduction of scale-space filtering in 1983 [15] the technique has been used in a variety of applications including catastrophe point detection [4], vortex tracking on time-dependent data [12], estimating image deformations [14], feature detection [13], and range image filtering [11]. Relatively fewer researchers [9, 10] have looked into 3D model (or mesh) simplification based on analysis at multiple scales. 3D visualization is an expanding area of multimedia research covering graphics, imaging and network transmission. With advances in laser scanning and digital imaging it is now possible to scan objects with super high resolution texture (surface image) and depth at various surface locations (connected into a mesh). For example, the Stanford bunny (Fig. 2, left) commonly used as a test object has 69,000 triangles. The Zoomage 3D scanner (Fig. 2, right) can produce texture of 200 mega pixels and meshes with over 2 million triangles. The trend in multimedia applications is to use more polygons in order to produce photo-realistic 3D scenes. However, a large number of polygons impose challenges in terms of storage, processing, rendering and transmission. Fortunately, high resolution detail is not necessary in all circumstances. In Fig.1, when the mesh is closer to the viewpoint, more polygons and finer texture can represent better detail, but when the object is further away, keeping the same number of polygons and high resolution texture does not increase visual fidelity. Rendering less polygons with lower resolution texture is sufficient without loosing significant details on the object. Both transmission and rendering time are saved, by reducing the texture resolution and number of polygons from 1800 to 180. Based on the texture and mesh quality to be transmitted, the bandwidth between a server and a client (viewing workstation) can be accurately monitored [5] and the quality can be adjusted to allow the best possible visualization given a time constraint.

Level-of-detail (LOD) [1,2,3] is a 3D-visualization topic dealing with efficient polygon meshing and texture mapping based on viewpoint. Since human perception is less sensitive to details on an object when it moves further away and gets smaller, it is inefficient to render the same number of polygons as when an object is, say, a few feet away. Therefore the objective of LOD is to represent areas of low perceptual importance with a few large triangles, and represent areas of high perceptual importance with many small triangles.

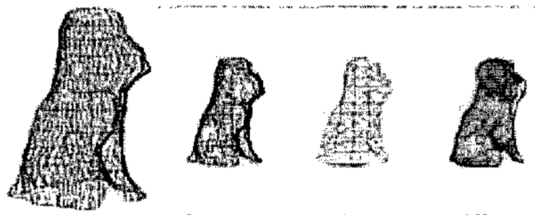
Fig.1: An example of a 1800 polygon 3D object at different distances (first two), and (third) mesh of same object using 180 polygons, texture mapped on 180 polygon object (fourth),

Fig.2: Stanford bunny (left) and Zoomage^{TM} 3D scanner from TelePhotogenics Inc. (right) used to capture 3D data.

A curvature equalization method is described by Scarlatos *et al.* [6]. They tried to balance the curvature of the input data within each triangle by adjusting the triangulation of the original surface. This work was developed for shape fitting, whereas we consider adaptive 3D multimedia transmission. Kalvin *et al.* used a simple patch decimation method in their efforts to create surface models from medical data [7], after an initial polygonal surface is created to approximate the input data, adjacent coplanar polygons are merged to simplify the model. Since only precisely coplanar faces are merged, the degree of simplification is largely dependent on the curvature of the object, and thus only limited simplification is obtained. Hinker *et al.* extended the patch decimation method to merge the nearly coplanar polygons [8]. If the angle between the normal vectors of two adjacent triangles is below a given bound error, the two triangles are merged. Finally, the merged polygons are re-triangulated with a simple and robust method. However, this method is highly ineffective for surfaces with high curvature.
Despite all the excellent research results in the areas of scale-space analysis and mesh simplification in the past two decades, one area that has received little attention is the relation between mesh and texture in the simplification and visualization process.

In recent research, we discussed an approach for variable compression of texture based on model complexity [16]; and discussed the relative importance of texture and mesh in human perception [17]. In the current paper we address the issue of taking the texture variations into account in simplifying a mesh. The work is based on the assumption that minor surface variation may be unimportant in areas where there is little variation in surface texture. Our approach for detecting small changes vs. major variations is based on scale-space filtering.

The rest of this paper is organized as follows: Section 2 explains the modified version of scale-space analysis. Section 3 discusses the scale map and fragment map. Section 4 summarizes the experimental results. Section 5 gives the conclusion and future work.

2. Scale-space filtering for 3D objects

Scale-space filtering (SSF) is based on analyzing the zero-crossings of a signal for varying scales of smoothing, of the signal. The advantage of using SSF is its ability to smooth locally or globally depending on the filter window size. Fig. 3 is an example of global smoothing using a window size of 201 on a sample space of 256. If a smaller window size is used, smoothing will converge before reaching the bottom scale.

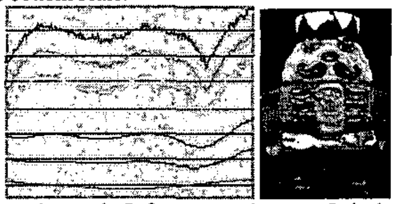

Fig.3 Increasing scale S_i from top to bottom. S_0 is the original signal extracted near the bottom of the Nutcracker model

The zero-crossings at different scales (Fig.4) can be realized directly by smoothing with the second derivative of the Gaussian (called Laplacian-of-Gaussian or LoG).

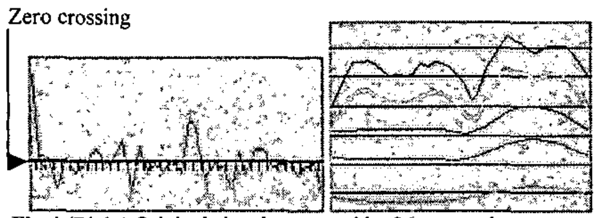

Fig.4 (Right) Original signal generated by 36 scan points extracted from the Nutcraker model, and four other smoothed scales, with 18, 8, 6, 4, 2 zero crossing respectively from top to bottom. (Left) 18 zero Crossing of the original signal

SSF in 2D can be summarized by the following equations:

$$w_G(x,y) = \begin{cases} \dfrac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x^2+y^2)}{(2\sigma^2)}} & (x,y) \in W \\ 0 & elsewhere \end{cases}$$

$$w_{LoG}(x,y) = \begin{cases} -\dfrac{1}{\pi\sigma^4}\left[1 - \dfrac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} & (x,y) \in W \\ 0 & elsewhere \end{cases}$$

$$fS(x,y) = \int_{-t}^{t}\int_{-t}^{t} f(x+u, y+v) w(u,v) du dv$$

Where w(x,y) represents the weight at pixel (x,y), *f* represents the original signal (image) and *fS* the smoothed image & the weights are assumed to be defined in a square window W of length 2t+1. In an implementation with an image we actually use summation instead of integrals, and normalize the Gaussian weights so that the sum of all the weights equals 1.

SPHERICAL APPROACH

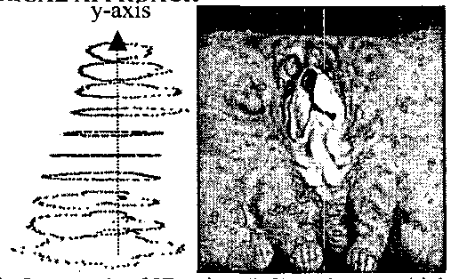

Fig.5 A sample of 3D points (left), and texture (right)

We achieve SSF of a 3D model as follows: First note that the data acquired (Figure 5) can be represented as $R_x(\alpha,Y)$; where α is the angle on a horizontal plane around the y-axis of rotation of an object, Y is the vertical location, and R_x denotes the distance to the surface of an object for a given (α,Y) pair. SSF for a 3D model is thus similar to a 2D image, for the simplified mesh representation considered here, with f(x,y) replaced by $R_x(\alpha,Y)$. Also, the appropriate scaling along the horizontal and vertical directions can be significantly different, depending on the variance of the sample points for a given region. Thus, the equations above need to be modified to:

$$w_G(\alpha,y) = \begin{cases} \dfrac{1}{\sqrt{2\pi\sigma_1\sigma_2}} e^{-\frac{\phi\alpha^2}{2\sigma_1^2}-\frac{\varphi y^2}{2\sigma_2^2}} & (\alpha,y) \in W \\ 0 & elsewhere \end{cases}$$

$$R_xS(\alpha,y) = \int_{-t}^{t}\int_{-t}^{t} R_x(\alpha+u, y+v) w(u,v) du dv$$

For uniform sample points, ϕ and φ equal 1, but for irregular sampling, ϕ and φ are used to accommodate the variable inter-sample distance. Note that in the actual implementation we use two passes of 1-D filters, since all the filters discussed above are separable. The vertices are first smoothed along the x-axis and then along the y-axis. Fig.6 shows the face features change towards a spherical surface going from low to high scales.

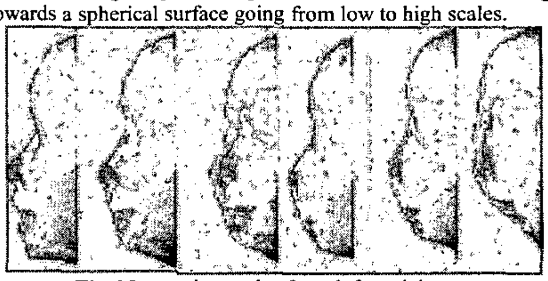

Fig.6 Increasing scales from left to right

For a 3D object, with full color real texture there are two components that can be filtered — the mesh and the texture. For color texture, three factors can be considered; *e.g.*, hue, saturation and intensity (HSI) representation. Saturation measures the amount of whiteness added to a pure color and the hue measures the tone of the color. From perceptual experiments done in the past by various researchers, it is well known that the human visual system (HVS) is most sensitive to intensity changes and is much less sensitive to hue and saturation changes. Most image compression standards (such as JPEG) reduce the resolution of the H & S components by half in the horizontal and vertical directions, thereby for every four 8 x 8 I-blocks in JPEG there are one 8 x 8 H-block and one 8 x 8 S-block. An analogous concept in SSF based reduction will be to consider different scales for smoothing and subsequently sub-sampling the H, S and I components. However, to simplify our initial implementation we only follow the JPEG standards in reducing texture.

Another issue that needs to be considered is: How to determine the relative importance of model *vs.* texture? In recent research [17] we found that improving model resolution improves perceptual quality following an exponential curve whereas enhancing texture makes perceptual quality increase linearly. Based on this finding and the fact that the texture component in a 3D image requires greater storage or bandwidth for transmission, we use SSF model analysis for texture reduction. The approach can be summarized as follows:

(a) Perform a SSF analysis of the model and identify regions of strong persistent structures vs. regions of small surface variations, at different scales. Generate a priority list of feature points from strong to weak persistence.

(b) Select a scale S_i based on viewing distance, pop feature points at scale S_i from priority list and distribute to their respective texture fragments.

(c) Based on bandwidth or storage limitation, say B, perform the following:

```
While (Size_Mesh + Size_Texture > B)
  {increase scales for mesh to next significant level;
   redistribute feature points at this scale $S_{i+1}$;
   compute uniform quality $Q_{i+1}$;
   For each texture fragment do
     {compute measure of complexity of mesh ($n\#F_{i+1}(x,y)$)
       If ($n\#F_{i+1}(x,y)$ == threshold) { quality = $Q_{i+1}$; }
       Else if ($n\#F_{i+1}(x,y) >$ threshold) { quality = $Q_{i+1} + \varepsilon$; }
       Else { quality = $Q_{i+1} - \varepsilon$;}}}
```

3. Integrated Texture and Mesh (TexMesh) Simplification

There have been extensive surveys on simplification methods in the last decade, and the research on 3D mesh simplification has come to maturity as suggested in [1]. However, previous discussions focused mainly on geometry without integrating real texture in a coherent approach. Many studies emphasized the importance of million of triangles in order to present fine surfaces, but ignored high resolution real texture which has been shown through user evaluations to have more impact on perceptual quality in 3D visualization [17]. In addition to storage, high resolution texture requires more processing, transmission and rendering resources. An integrated approach, based on Scale Map and Fragment Map, is proposed in this paper to achieve geometry and texture simplification. During on-line visualization, statistics collected through preprocessing are used for efficient rendering of feature points.

Feature Point Determination

In this paper, feature points are defined as a set of vertices which can best represent the geometry of a 3D model, given a constraint on the number of vertices. For example, given the constraint 8, the 8 vertices of a cube are feature points. At any scale S_i, feature points are detected by applying LoG. Vertices creating zero crossing are recorded as feature points. Based on scale-space theory, the number of feature points (structures in the sample space) diminishes when i approaches infinity. This concept best describes the simplified geometry of a 3D model moving away from the viewer, where feature points become increasingly invisible.

Scale Map

Three-dimensional (3D) vertices are sorted and assigned a unique id L, *i.e.*, $0 \le L < N$, based on their y then x coordinates. A Scale Map is a 2D display of all 3D vertices in a matrix with rows and columns corresponding to the y and x values respectively. The default value for each vertex is 0 corresponding to scale level 0. At each scale S_i only feature points of that scale are updated with the value i. During preprocessing, Gaussian filters with increasing sigma values σ_i are used from scale S_0 to S_{max}, *i.e.*, $0 \le i \le max$, where S_{max}, corresponding to scale at infinity. Zero crossings are detected from the filtered space G_i where G_0 represents the set of original unfiltered range data. By adding feature points from high to low values, meshes can be constructed from coarse to fine. The final scale map can be implemented as a priority queue, where the next required vertex is popped from the queue at constant time. Fig.7 shows an example of how feature points vary at different scales.

Scale i	0	1	5	10	25
# of features	1872	1267	1110	969	322

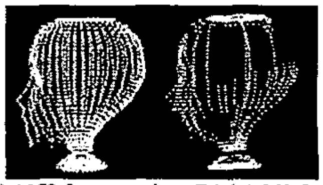

Fig.7 (Left) 1872 feature points (Right) 969 feature points

Fragment Map

The texture of a 3D model is fragmented into numX*numY equal pieces. NumX and numY are determined by dividing the width and height of the texture by the width and height of a fragment respectively. To apply JPEG compression efficiently, keeping in mind the size of macroblocks, the width and height of a fragment is chosen as a multiple of 16. The entire texture is also adjusted so that there is no partial fragment. Similar to the scale map, the fragments are arranged in a matrix with numY rows and numX columns. Since each 3D vertex is associated with a 2D texel, it is possible to distribute the vertices into the numX*numY fragments. For example, a texture image with dimension 4800*1600 pixels, can be divided into 7,500 fragments of size 32*32 pixels. Experimental results show that the sum of the fragments is less than the size of the combined JPEG file. Individual fragments can be transmitted to the client site for recombining and rendering. This fragmented approach is most effective over distributed networks.

Since the HVS is less sensitive to detail far away, the texture quality Q_i at each scale S_i needs to increase only when i decreases. Depending on the viewing distance, the corresponding S_i and Q_i are selected. Instead of applying uniform quality to all fragments, we use an adaptive approach so that texture quality of each fragment (x,y) varies depending on how many feature points are associated with it. At a given scale i, we define $\#F_i(x,y)_{max}$ as the maximum # of feature points per fragment, and $\#F_i(x,y)_{min}$ as the minimum # of feature points per fragment. $\# F_i(x,y)$ is then normalized:

$$n\# F_i(x,y) = \frac{\# F_i(x,y) - \# F_i(x,y)_{min}}{\# F_i(x,y)_{max} - \# F_i(x,y)_{min}}$$

The texture quality for fragment $Q_i(x,y)$ is computed as:

$$Q_i(x,y) = Q_i + (n\# F_i(x,y) - \Gamma)\,\Delta Q$$

where threshold $\Gamma = n\#F_{imean} \in [0,1]$ is the mean of the normalized values $n\#F_i(x,y)$. ΔQ controls the deviation (+/-) from Q_i. In other words, more feature surface is displayed with higher quality and less feature surface is displayed with lower quality texture. Γ can be adjusted, depending on the available bandwidth. If $\Gamma > n\#F_{imean}$, texture fragment quality and size will decrease. If $\Gamma < n\#F_{imean}$, quality and size will increase.

4. Experimental Results

We use the Intel JPEG Library compression algorithm. Preprocessing is implemented in C, and 3D display is implemented in Java3D. Two texture resolutions (A) 1024^2 & (B) 256^2 pixels are used for the dog model with 256 & 64 fragments respectively. The total size of fragments for uniform and variable quality are show below:

Dog texture	Scale i	Uniform (KB)	Variable (KB)
(A) 1024^2	10	249	215
	20	200	188
(B) 256^2	10	47	50
	20	42	42

Since the compressed size varies depending on the texture pattern, we tested out different textures (Fig.8) on the mesh:

Fig.8 Texture of Goddess, Nutcracker, Head & Stone Vase

Texture Pattern	Scale i	Uniform Quality (KB)	Variable Quality (KB)
Goddess(A)	10	293	240
Goddess(B)	20	44	43
Nutcracker(A)	10	222	200
Head(A)	10	253	214
Stone Vase(A)	10	289	243

The results show that the variable quality approach generates better perceptual display without sacrificing bandwidth. A comparison of uniform and variable texture quality 3D mapping is shown in Fig.9. The eye fragment is a high feature surface, which is displayed (scale i=25 with 1471 feature points) at quality 50 by the variable approach, and only quality 10 by the uniform approach. Total texture fragment size is 41 KB in both cases.

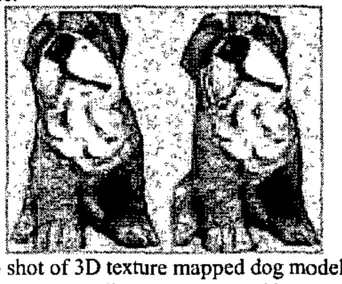

Fig.9 Snap shot of 3D texture mapped dog model showing (left) variable quality and (right) uniform quality.

5. Conclusion and Future Work

An integrated TexMesh simplification based on scale-space analysis, using scale map and fragment map, is proposed in this paper. We apply LoG to generate a feature point priority queue for efficient extraction of feature points at different scales. Since statistics are collected during preprocessing, feature points can be extracted at constant time in an on-line application. Based on how many feature points appear in each texture region, variable quality compression is applied to fragments. Experimental results show that variable qualities, giving less simplified surfaces higher quality, provide better perception on 3D objects and make more efficient use of bandwidth. In future work, we will evaluate the perceptual quality of 3D objects generated from and performance of our TexMesh model. We will experiment with more 3D models and analyze the relation between perceptual quality and ΔQ. Currently we are also looking into the effect of other parameters, *e.g.*, color components in a texture, on perceptual quality.

6. References

[1] D. P. Luebke, "A Developer's Survey of Polygonal Simplification Algorithms", IEEE Computer Graphics and Applications, May/June 2001 p24-35

[2] J.C.Xia et al., "Adaptive Real-Time Level-of-detail-based Rendering for Polygonal Models" IEEE Trans. on Visualization and Computer Graphics, June 1997.

[3] H.Hoppe, "Progressive meshes" Proceedings of SIGGRAPH 1996, L.A. pp. 99-108

[4] A. Kuijper et al., "The application of catastrophe theory to image analysis, Technical Report, 2001.

[5] Y. Yu, I. Cheng and A. Basu, "Optimal adaptive bandwidth monitoring," IEEE Trans. on Multimedia, September, 2003.

[6] L. L. Scarlatos and T. Pavlidis, "Optimizing triangulations by curvature equalization," In *Proc. Visualization'92*, pp. 333–339. 1992.

[7] A.D. Kalvin et al., "Constructing topologically connected surfaces for the comprehensive analysis of 3D medical structures," *SPIE Vol. 14*, pages 247–259, 1991.

[8] P. Hinker and C. Hansen, "Geometric optimization," In *Proc. Visualization '93*, pages 189–195, San Jose, CA, October 1993.

[9] M. Pauly et al., "Multi-scale feature extraction on point-sampled surfaces," Eurographics 2003, Granada, Spain.

[10] L.P.Kobbelt et al., "Feature sensitive surface extraction from volume data," SIGGRAPH 2001.

[11] P.Boulanger et al., "Intrinsic filtering of range images using a physically based noise model," Vision Interface 2002, Calgary, Canada.

[12] D. Bauer and R. Peikert, "Vortex tracking in scale-space," Eurographics-IEEE TCVG Symposium on Visualization, 2002.

[13] T. Lindberg, "Feature detection with automatic scale selection," International Journal of Computer Vision, vol. 30, no. 2, 1998.

[14] T. Lindberg, "A scale selection principle for estimating image deformations," International Coference on Computer Vision, Cambridge, MA, USA, 1995, pp. 134-141.

[15] A.P. Witkin, "Scale-space filtering," International Joint Conference on AI, 1983, pp. 1019-1022.

[16] I. Cheng, "Efficient 3D Object Simplification and Fragmented Texture Scaling for Online Visualization," IEEE International Conference on Multimedia & Expo, Baltimore, USA, 2003.

[17] Y. Pan, I. Cheng, and A. Basu, "Quantitative metric for estimating perceptual quality of 3D objects," IEEE Transactions on Multimedia, to appear.