Adaptive Online Transmission of 3D TexMesh Using Scale-Space Analysis

Irene Cheng[*] *and Pierre Boulanger*

Department of Computing Science, Univ. of Alberta, Edmonton, CANADA, {lin,pierreb}@cs.ualberta.ca

Abstract

Efficient online visualization of 3D mesh and photo realistic texture is essential for a variety of applications, such as museum exhibits and medical images. In these applications synthetic texture and a predefined set of views is not an option. We propose using a mesh simplification algorithm based on scale-space analysis of the feature point distribution, combined with an associated analysis of the surface texture, to address the needs of adaptive online transmission of high quality 3D objects. The premise of the proposed textured mesh (TexMesh) simplification is the following: minor variations in texture can be ignored in relatively smooth regions of a 3D surface, without significantly affecting human perception. Statistics on 3D feature point distribution and their associated texture fragments are gathered during preprocessing. Online transmission is based on these statistics, which can be retrieved in constant time. Based on monitored bandwidth a scaled mesh is first transmitted. Starting from a default texture quality, we apply an efficient Harmonic Time Compensation Algorithm based on the current bandwidth and a time limit, to adaptively adjust the texture quality of the next fragment. Properties of the algorithm are proved. Experimental results show the usefulness of our approach.

Keywords: 3D transmission, Scale-space analysis, Bandwidth monitoring, Time-compensation algorithm

1. Introduction

Efficient bandwidth utilization and optimal transmission quality are among the main objectives when transmitting 3D models. Since mesh data is usually small compared with texture data, our focus is to adapt the texture quality to the current bandwidth and a specified time. A historic average can be used to estimate current bandwidth [6], but this approach can cause unacceptable over or under estimation because of bandwidth fluctuations. An optimal bandwidth monitoring approach can provide a more accurate estimation by sacrificing a portion of the transmission time [28]. In this paper, we propose an adaptive approach, which does not need to sacrifice transmission time for bandwidth estimation, while efficiently adjusting the quality of the fragments not yet transmitted.

In order to optimize bandwidth, a multi-scale incremental simplification approach is most suitable for online applications. Multi-scale allows a coarse version to be refined incrementally. However, if a simplification process involves relocation of mesh vertices or texture coordinates between scales, then an entirely new version, instead of the vertices added to the previous version, has to be transmitted [4, 11, 21, 25]. In the progressive meshes method, although the original mesh can be recovered exactly after all data are received, the edge collapse transformation creates new vertices and the *vsplit* record stream increases network workload [13]. Xia's adaptive real-time LOD technique also involves vertex relocation [27]. We apply vertex removal and hole filling without affecting the 3D point and texel coordinates.

In recent years, researchers started to incorporate color and texture into their mesh simplification models. When texture is mentioned in the literature, it often refers to synthetic or animated texture [24]. Synthetic texture can be estimated. For example, when walking through an animated scene, the next frame can be predicted based on available neighboring data [8]. Experimental results show that this technique has better quality and higher compression factor than MPEG. An image-driven simplification method is used to display textures using images from multiple views [17]. However, rendering the entire model for every edge in every viewpoint for different scales is expensive, even with hardware-accelerated rendering. The high-resolution texture used in our TexMesh model is different from the per pixel color stored in each vertex [10,12,22,23]. For applications requiring real life texture, interpolating colors between vertices is not acceptable. The non-interpolated texture we use has resolution much higher than the mesh. It was observed in perceptual experiments that the human visual system is more sensitive to higher texture resolution after the mesh reaches an optimal density [20]. Our TexMesh model uses photo-realistic texture images, with resolution up to millions of pixels, suitable for displaying on small monitors or high definition screens in reality centers. Photo-texture is used in compressed texture maps [29], but their effort is on recovering geometry from texture patches retrieved from multiple photographs. A distance-based technique is applied to photo-textured terrain [16]; however, color interpolation between pixels is necessary to avoid blocky appearance of terrain texture.

In [19], the joint geometry/texture progressive coding method applies wavelet transform to encode the mesh and texture data for transmission, but the method is not adaptive to fluctuating bandwidth. Wavelets were also used to create space optimized texture maps, which did not require any on-chip compression support [2]. In our method, we apply scale-space filtering and zero-crossing detection to extract feature points. Each scale is associated with a default texture quality, and the quality of each fragment is allowed to

[*] The support of an NSERC Ph.D. Scholarship is gratefully acknowledged.

deviate within a limit based on the number of feature points in it. The quality is later readjusted according to current bandwidth. By relating each scale to a viewing distance, automatic selection of a corresponding simplified textured mesh is possible. Another advantage of using scale-space analysis is its capability of handling both local and global smoothing, which is controlled by the filter window size and parameter σ. Many simplification techniques are restricted to smoothing in the local neighborhood. For example, the simplification envelopes algorithm [9] is based on the principle that each vertex is displaced within ±ε to the extent constrained by surface curvature. Avoidance of self-intersections prevents drastic simplification.

Ever since the introduction of scale-space filtering in 1983 the technique has aroused intense research interest [26], but previous applications focused on image deformation, vortex tracking, noise elimination and segmentation in 2D images [2, 3, 14, 15]. As far as we know, scale-space filtering has not been used in the literature to compute feature point distribution, and thereby determine texture quality adaptively. In recent research, we discussed an approach for variable compression of texture based on model complexity [5]; and discussed the relative importance of texture and mesh in human perception [20]. In the current paper we address the issue of adaptive quality adjustment taking into account bandwidth fluctuations within a given time period. The work is based on the observation that texture quality is more important in regions with high feature density. Our technique for detecting small changes *vs*. major variations is based on scale-space filtering. Note that one major difference between wavelets and our scale-space approach is that wavelets scale up or down by a factor of 2; by contrast, in scale-space the scales at which changes occur are object dependent and not fixed beforehand.

In this work, we extend our basic textured mesh (TexMesh) model to incorporate the adaptive capability to transmit 3D objects. We use scale-space analysis and zero-crossing detection to extract feature points at different levels of details (LOD). A scale map and a fragment map are generated based on feature point distribution. The TexMesh model applies a dynamic strategy and adapts to the current bandwidth when computing texture quality of the next transmitted fragment. We apply a Harmonic Time Compensation Algorithm to ensure optimal use of the time limit and bandwidth. This approach supports efficient online transmission. By splitting the texture into fragments, distributed transmission and parallel processing is possible.

The remainder of this paper is organized as follows: Section 2 overviews the online adaptive strategy. Section 3 gives a summary of our basic TexMesh model, explaining how to apply a modified version of the Gaussian filtering and Laplacian, to extract feature points. Section 4 explains fragment map generation, and how texture quality is assigned to each fragment. Section 5 analyzes the adaptive transmission strategy. Section 6 concludes the work and outlines future directions.

2. Overview of Adaptive Strategy

Our strategy for adaptive online transmission of 3D objects has several components, which are shown in Figure 1.

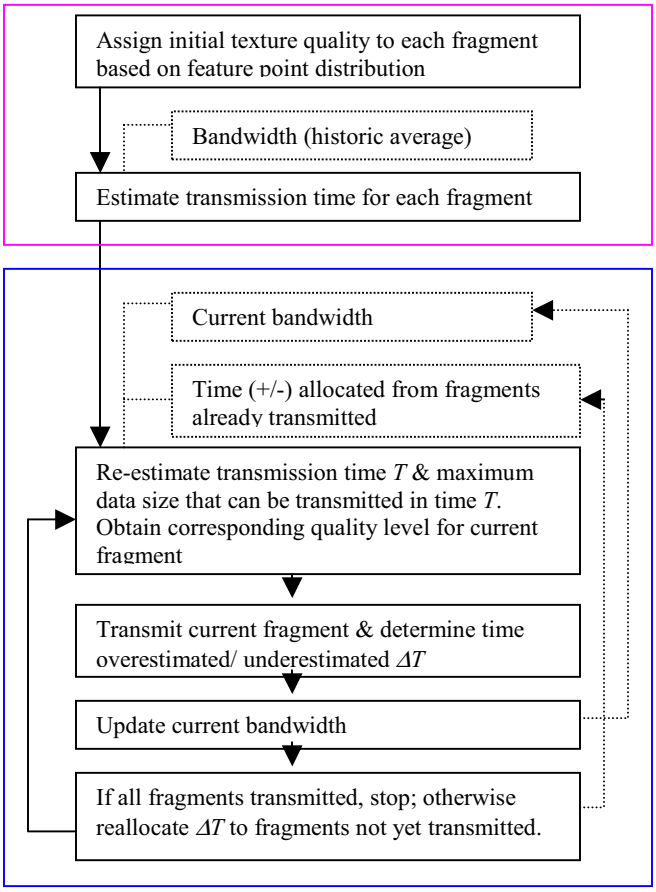

Figure 1: Summary of the adaptive online transmission strategy.

The components are divided into two groups: preprocessing is described in the upper group, and the online process is described in the lower group. The feature point distribution is a value ∈ [0,1], which is mapped onto a compression scale. For example, in the current implementation, we use the JPEG compression scale [0,100]. While wavelet coding applies to the entire image and is geometry-independent, our approach supports variable quality determined by the density of surface structures. Note that we use JPEG for convenience and wide support on the web and in JAVA; however, standards such as JPEG2000 can be used as well in the future to code fragments.

3. Basic TexMesh model

In this paper, we extend our basic model to address the issue of online adaptive transmission. A summary of the basic model is given in this section for completeness. Interested readers can refer to [7] for additional detail.

3.1 Level of details (LOD) in scale-space

The Gaussian filter is an efficient smoothing tool in computer vision. However, scale-space filtering had mainly been applied in 2D images and only recently has this

technique been used in computer graphics, with limited applications in 3D visualization. We use scale-space filtering (SSF) for adaptive on-line 3D TexMesh simplification and transmission. Traversal between the different scales, or LOD is achieved by varying the standard deviation parameter σ; the higher the value of σ the more is the smoothing. SSF is based on locating the zero-crossings of a signal at multiple scales. Zero-crossings can be used to detect the degree of persistence of a structure (feature) in a 3D model. Minor structures tend to diminish as σ increases, and only major structures survive at higher scales. When using a small window size, SSF eliminates signal noise in the local region. By using a bigger window size, the filtering or averaging effect covers a larger surface. Fig. 2 is an example of global smoothing using a window size of 201 on a signal of 256 values. To obtain global smoothing, the window size has to be at least twice the standard deviation computed from the sample space (covering at least 97.7% of the sample data in a normal distribution). If a smaller window size is used, smoothing will be restricted and converge before reaching the bottom scale in Fig. 2.

Theoretically, 100% global smoothing will end up with a monotonous surface losing all the surface features. When considering the human visual system, this is not necessary because human vision is insensitive to details beyond a certain distance. For a perceivable object, the filter window can be smaller than twice the standard deviation to save computation time. In our experiments, we applied a window size of 1.4 times the standard deviation. We found that this window size provides sufficient simplification for objects placed at a distance close to infinity in the virtual world. Further simplification beyond this point by using a bigger window is not necessary. Perceptual evaluation is not the focus of this paper, and will be discussed in future work.

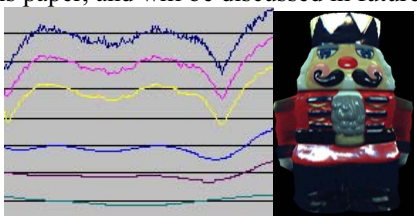

Fig. 2: Increasing scale S_i from top to bottom. S_0 is the original signal extracted near the bottom of the Nutcracker 360 scan-points (per row) model. Note that local variations (fine details) in the original signal are gradually removed and the scaled signal becomes smoother.

The zero-crossings at different scales can be computed by applying the second derivative of the Gaussian (called Laplacian-of-Gaussian or LoG). 18 feature points are identified in the original signal (Fig. 3, right). By increasing σ, the number of feature points decreases from 18 to 2 as reflected by the increasing smoothness of the scaled values (Fig. 3, left).

3.2 Spherical approach on scanned range data

Modern laser scanners detect depths and generate 3D vertices in the form of point clouds. Fig. 4 (left) shows a 6-inch dog model. The generated point cloud (Fig. 4 middle) is then mapped with the scanned texture (Fig. 4 right) to generate the texture mapped 3D model.

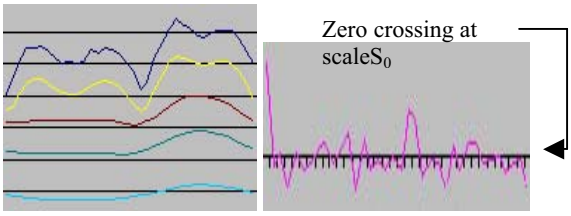

Fig. 3: (Left) The top is the original signal with 18 zero crossings, generated by 36 scan points extracted from the Nutcraker model. The other four smoothed scales have 8, 6, 4, and 2 zero crossing respectively from top to bottom. (Right) 18 zero crossings detected in the original signal S_0.

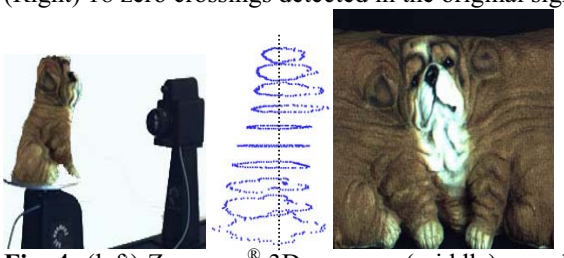

Fig. 4: (left) Zoomage® 3D scanner, (middle) sample of 3D points, and (right) scanned texture.

We achieve SSF of a 3D model as follows: First note that the data acquired (Fig. 4 middle) 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 $I(x,y)$, for the simplified mesh representation considered here, with $I(x,y)$ 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, SSF in 3D can be summarized by the following equations:

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

$$Rx * S(\alpha,y)=\int_{-t}^{t}\int_{-t}^{t} Rx(\alpha+u,y+v)w(u,v)dudv \quad (2)$$

$$w_{LoG}(\alpha,y)=\begin{cases}-\dfrac{1}{\pi\sigma^4}\left[1-\dfrac{\phi\alpha^2+\varphi y^2}{2\sigma^2}\right]e^{\frac{\phi\alpha^2+\varphi y^2}{2\sigma^2}} & (\alpha,y)\in W \\ 0 & elsewhere\end{cases} \quad (3)$$

Here $w_G(\alpha,y)$ represents the weight at pixel (α,y), R_x*S represents the smoothed image, and w_{LoG} is the Laplacian of Gaussian function. The weights are defined in a square window W of length $2t+1$. In the discrete case, *e.g.* with a real image, we actually use summation instead of integrals,

and normalize the Gaussian weights so that the sum of all the weights equals 1.

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 the resulting values are filtered once more along the *y*-axis. Fig. 5 shows the face features, of a head model, change towards a smoother spherical surface when going from low to high scales (left to right). The original mesh contains 1,872 vertices and 3,672 faces. The other five meshes are generated at increasing scales as follows:

Scale	# of vertices removed	# of faces in mesh
5	685	2226
7	744	2108
10	824	1948
15	985	1624
20	1196	1190

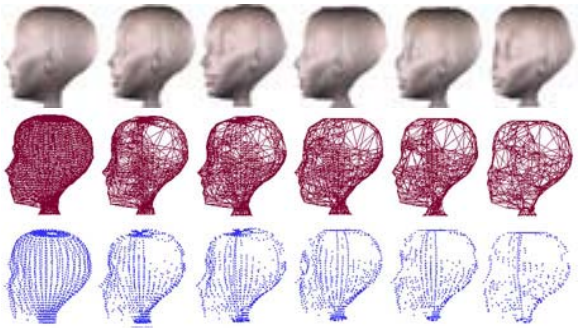

Fig. 5: Increasing scales from left to right (Top) 3D mesh with texture, (Middle) 3D mesh and (Bottom) feature points extracted at scale S_i.

4. Variable texture quality based on feature point distribution

In [18] it was suggested that 3D mesh simplification has come to maturity. However, previous simplification methods focused mainly on geometry without integrating real texture in a coherent approach. Many studies emphasized the importance of millions 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 [20]. Adaptive texture quality is essential for online transmission. In our approach, statistics on feature points collected through preprocessing are used for efficient transmission of geometry and texture data given limited network resources.

Feature points are defined as a set of vertices, which can best represent the geometry of a 3D model. In Fig. 5, for example, the original head model contains 1872 feature points (scale S_0). After removing 1196 vertices, it is represented by 676 feature points at scale S_{20}. At any scale S_i, feature points are detected by applying LoG. Vertices creating zero crossing are recorded as feature points and assigned the value *i*. Each feature point is then represented by three components: *(i, (tx,ty), (gx,gy,gz))*.
The second and third components are the 2D texture and 3D vertex coordinates, respectively. Based on scale-space theory, the number of feature points (structures in the sample space) decreases as scale level increases. This concept best describes how objects are perceived by the human visual system when they move from close to far.

4.1 Fragment map and geometry-driven texture quality estimation

The texture image of a 3D model can be transmitted as one block or a collection of sub-blocks. The advantage of using sub-blocks is to facilitate distributed transmission and applying variable qualities to different texture regions as explained below. The main concern is whether the additional headers and meta-data will increase the overall volume of data that needs to be transmitted. In this section, we will show that sub-dividing into smaller blocks of optimal dimension does not increase the overall volume for high-resolution texture images. Instead, the sub-block approach helps to fully utilize the available bandwidth.

The texture image is fragmented into N_X*N_Y equal pieces after determining the optimal size of a fragment. To apply JPEG compression efficiently, keeping in mind the size of macro-blocks, the optimal dimension of a fragment is chosen as a multiple of 16. The entire texture is also adjusted so that there is no partial fragment. For example, a texture image with dimension 4800*1600 pixels, can be divided into 7,500 fragments of size 32*32 pixels. Fragments are arranged in a matrix with N_Y rows and N_X columns. Since each 3D vertex is associated with a 2D texel, it is possible to distribute the vertices into the N_X*N_Y fragments. We used five texture patterns (Fig. 6) to compare the fragmented and non-fragmented size in different qualities using the Intel JPEG compression library. Each fragment has a dimension of 16*16 pixels. Experimental results show that the sum of the fragments is significantly less than the size of the non-fragmented JPEG file for images of dimension greater than 256 pixels. For high-resolution images, it is therefore advantageous to transmit individual fragments to the client site before recombining and rendering. This fragmented approach is also suitable for distributed network retrieval.

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

Fig. 6: Texture images used in experiments to show that sum of sub-blocks of optimal size is significantly less than the size of non-fragmented image for high resolution texture.

Since the human visual system is less sensitive to details far away, the texture quality Q_i at each scale S_i needs to increase only when *i* decreases. In other words, display quality is inversely proportional to viewing distance. Given

a viewing distance, the corresponding S_i and Q_i are selected. Instead of applying uniform quality to all fragments, we use a variable approach so that texture quality of each fragment *(x,y)* varies depending on the number of feature points associated with it. We illustrate how the distribution of feature points relates to the quality of texture as follows: In Fig. 8, the grenade has vertical structures on the surface, and therefore the feature point distribution is higher than the back of the nutcracker, which is comparatively flat. Note that even if the texture quality is reduced to half, there is no significant perceptual degradation on the nutcracker (Fig. 7). However, the grenade on the right (Fig. 8) shows noticeably lower perceptual quality. Based on this finding we adopt a variable approach by applying different qualities on texture fragments depending on the feature point distribution, instead of applying a fixed quality to all fragments. Furthermore, the variable qualities are computed adaptively based on the current bandwidth. An adaptive approach is necessary when transmitting data on the Internet because bandwidth fluctuates and can adversely affect the expected Quality of Service (QoS).

Fig. 7: A snap shot of the rear-view of nutcracker: (left) original texture quality, and (right) half of the original texture quality.

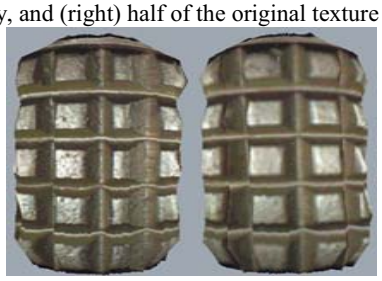

Fig. 8: A snap shot of the grenade 3D model: (left) original texture quality, and (right) half of the original texture quality.

Before discussing the adaptive approach, we first explain how variable qualities are assigned to different fragments.

Let *(x ,y)* be the coordinates in the N_X*N_Y fragment map, and $f_i(x, y)$ be the # of feature points in fragment *(x,y)* at scale S_i.
At a given scale *i*, $f_i(x, y)$ is normalized to:

$$\eta_i(x,y) = \frac{f_i(x,y) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} \quad (4)$$

$f_i^{\max}$ and $f_i^{\min}$ are respectively the maximum and minimum # of feature points per fragment at scale S_i.
The texture quality of fragment *(x,y)* at scale S_i is computed as:

$$Q_i + (\eta_i(x,y) - \Gamma)\,\Delta Q \quad (5)$$

where:
ΔQ: Quality tolerance limit for each scale controlled by an upper and a lower bound; analogous to the depth of field in photography. ΔQ is the tolerance range when displaying 3D objects at a given distance. Given a viewing distance, the human visual system finds this range of qualities satisfactory.
Q_i: Default quality assigned to scale S_i.

In the current implementation, threshold $\Gamma \in [0,1]$ is the average of $\eta_i(x, y)$. Fragment *(x,y)* is assigned quality Q_i if $\eta_i(x, y) = \Gamma$. ΔQ controls the deviation (+/-) from Q_i. Regions on the 3D model surface with more feature points are displayed with higher quality, and less populated regions are displayed with lower quality. The overall texture quality, along with the data size, can be adjusted by changing Γ.

Let D_i be the total data size of all fragments at S_i. For each model texture, a lookup table is used to record D_i, and also the size and quality of individual fragments, for a range of Γ. Given a time limit and current bandwidth, the appropriate D_i, and the associated fragments are selected. The actual data size and quality transmitted will change according to the current bandwidth. This adaptive strategy is discussed in the next section.

5. Adaptive bandwidth monitoring and texture quality determination

Because of bandwidth fluctuation, current bandwidth has to be monitored periodically in order to maintain a good estimate of the data size that can be transmitted in a specified time T_0. To minimize the discrepancy, we reallocate the time surplus/deficit to the fragments not yet transmitted.

The $n = N_X$*N_Y fragments are pre-sorted in decreasing $\eta_i(x, y)$ values, *i.e.*, from 1 to 0,

$$F_{list} = \{ F_1, ..., \bar{F}, ..., F_n \}, \ i.e., \ \bar{F} \text{ has quality } Q_i.$$

The first fragment to be transmitted is $\bar{F}$ with quality Q_i.

Based on a time limit T_0 and a historic bandwidth average β_0, we estimate maximum data size to be transmitted as:

$$D_1 = T_0 * \beta_0$$

Where:
β_k is the current bandwidth (KB/sec.) recorded after k fragments are transmitted, *i.e.*, $0 \le k < n$. β_0 is the historic average bandwidth before transmission.
T_k is the time left after k fragments are transmitted. T_0 is the original time limit (seconds) specified, and
D_{k+1} is the maximum data size that can be transmitted given β_k and T_k.

The fragment list F_{list}, best matching D_1, is selected from the lookup table. Size of $\bar{F}$ is used to estimate the transmission time of the first fragment:

$$est_1 = T_0 * \frac{d_1}{D_1} \text{ , or } est_1 = \frac{d_1}{\beta_0} \quad (6)$$

Where: d_k represents the data size of the k^{th} fragments,
est_k : The estimated time required to transmit fragment *k*

We estimate the transmission time est_g for all the remaining fragments, *i.e.*, $2 \le g \le n$:

$$est_g = T_0 * \frac{d_g}{D_1} \quad (7)$$

After d_1 is transmitted, we have the updated bandwidth β_1 based on the time act_1 recorded when transmitting d_1 :

$$\beta_1 = \frac{d_1}{act_1} \quad (8)$$

Where act_k is the actual time needed to transmit the fragment *k*.

The next fragment is selected as follows:
(a) The leftmost fragment in F_{list} if $\beta_1 \le \beta_0$, and
(b) The rightmost fragment in F_{list} if $\beta_1 > \beta_0$

Let: ΔT_k = Difference between estimated and actual transmission time for k^{th} fragment; *i.e.*, $est_k - act_k$
Δt_k be the cumulated compensating time (+/-) allocated to the k^{th} fragment from the previous *k*-1 fragments (refer to Algorithm 1 below), and
wt_f be the weight applied to the f^{th} fragment when allocating ΔT_{k-1}, *i.e.*, $k \le f \le n$.
(a) If the actual bandwidth is lower than the estimated one, loss of time ΔT_1 has to be compensated when transmitting the remaining *n*-1 fragments, so that each remaining fragment has to share a portion of ΔT_1. Instead of the initial est_2 computed in Equation (7), the 2^{nd} fragment has $wt_2 * \Delta T_1$ seconds less, where wt_2 is the assigned weight. We regain the time by transmitting the leftmost fragment in F_{list} with reduced quality.

(b) Similarly, if the actual bandwidth is greater than the estimated one, the gained time ΔT_1 is allocated to the remaining *n*-1 fragments, so that each remaining fragment can have additional time. Instead of the initial est_2 , the 2^{nd} fragment has $est_2 + wt_2 * \Delta T_1$ seconds. We adjust the time by transmitting the rightmost fragment in F_{list} with increased quality.

Based on the revised est_2 , we compute: $d_2 = \beta_1 * est_2$; and then obtain corresponding quality for the 2^{nd} fragment from the lookup table using d_2 . In general, after *k*-1 fragments are transmitted:

$\Delta T_{k-1} = est_{k-1} - act_{k-1}$ (9) $\Delta t_k = \Delta t_k + \Delta T_{k-1} * wt_k$ (10)

The computation of weight wt_k is explained in Algorithm 1.

$est_k = est_g + \Delta t_k$ (11)

$\beta_{k-1} = \frac{d_{k-1}}{act_{k-1}}$ (12), and, $d_k = \beta_{k-1} * est_k$ (13)

The quality for the k^{th} fragment is obtained from the lookup table based on d_k .
Since bandwidth fluctuation has a larger impact on the quality if ΔT_k has to be shared by a smaller number of fragments, fragments with quality $\cong Q_i$ are transmitted last to allow more flexibility for adjustment within the control limit ΔQ. Once the transmission is started, the quality of a fragment is self-adjusted depending on the updated bandwidth.

5.1 Harmonic Time Compensation Algorithm

Since later fragments have to share all preceding allocations, Algorithm 1 assigns decreasing weights $(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots..)$ to fragments $(k+1)^{th}$ to n^{th}, when reallocating ΔT_k .

Algorithm 1 – Harmonic Time Compensation
After transmitting k^{th} fragment,
div=2;

$\varsigma_k = \sum_{j=2}^{n-k+1} \frac{1}{j} \approx \ln(n-k+1)$;

for ($i = k+1; i \le n; i$++) {

$wt_i = \frac{1}{div * \varsigma_k}$;

$\Delta t_i += \Delta T_k * wt_i$; // allocate to remaining fragments

div^{++};

}

There are two questions we have to address:

(1) How efficient is the algorithm with respect to bandwidth optimization in a given time, and
(2) How does the adaptive approach affect the perceptual quality.

To prove the efficiency of the algorithm, we define Π as the time surplus/deficiency with respect to the limit T_0. Π is composed of three errors: estimation E_{est}, allocation E_{alloc}, and compensation E_{comp} errors. In Theorem 1 we establish the upper and lower bound of Π (Proof: See Appendix A).

Theorem 1: Π is bounded by:
$\Delta T_n + (\Delta T_{n-1} / 2) + (1.088 + \ln|\ln(n)|)\Lambda$
Where Λ is defined as the average difference between the estimated and actual transmission time for *n*-1 fragments,
i.e. $\Lambda = \frac{\sum_{j=1}^{n-1} \Delta T_j}{n-1}$.

The upper and lower bounds in Theorem 1 are verified by experimental results in the next section. We will show that our adaptive approach does not have an adverse effect on perceptual quality for reasonable bandwidth fluctuation.

5.2 Experimental results

Let n=256. Applying Theorem 1, we obtain:

$$E_{comp} \le 2.8\Lambda \text{ if } \Lambda \ge 0, \text{ and } E_{comp} \ge 2.8\Lambda \text{ if } \Lambda < 0$$

Since Λ is the average deviation over the entire transmission period, it is expected to be small. The other two components of Π: estimation error E_{est} (ΔT_n) and allocation error E_{alloc} ($\Delta T_{n-1}/2$), can be minimized by using sufficiently small data size for the last two fragments.

In order to see how Π responds to bandwidth fluctuation, we implemented a bandwidth monitor, and extracted three sets of bandwidths from an Ethernet connection on different days and different times. We then vary the value of β_0 below and above the average of the sample set within a reasonable range. The test file is 418KB with 256 fragments.

Bandwidth sample set	Actual bandwidth avg. (KB/sec)
1	41.68
2	45.64
3	42.07

β_0 (KB/sec)	Λ (sec)	Surplus/deficit (+/-)% of limit	Time Limit (sec)	Static %
20	0.038	0.399	20.9	52.0
23	0.031	0.346	18.17	44.8
26	0.023	0.280	16.07	37.6
29	0.015	0.195	14.41	30.4
32	0.011	0.142	13.06	23.2
35	0.003	0.010	11.94	16.0
38	-0.001	0.010	11	8.8
41	-0.004	-0.027	10.19	1.6
44	-0.008	-0.034	9.5	-5.5
47	-0.008	-0.058	8.89	-12.0
50	-0.012	-0.058	8.36	-19.8
53	-0.012	-0.090	7.88	-27.2

Fig. 9: Experimental results show that the Harmonic Time Compensation Algorithm has less than 1% deviation for a given time limit. β_0 is used for initial bandwidth estimation (Sample set 1).

One can observe that Π is minimum, when Λ is close to zero. Similar trends were obtained from Samples 2 and 3. By keeping the n^{th} and $(n\text{-}1)^{th}$ fragments sufficiently small, as in our experiments, the deviation from the time limit is within 1%. For comparison, the last column shows the discrepancy in percentage of time limit, should historic average bandwidth be used in a static approach.

To see how variable quality affects the overall visualization, we used β_0 = 32 and 50, together with a sample average of 41.68 (KB/sec), and applied to the dog texture. The original texture has quality Q_i = 80% and ΔQ_i is [40%, 100%]. Fig. 10 shows that the perceptual quality is maintained, after applying variable qualities to fragments adaptively in order to satisfy the time limit. Given the estimated texture in the middle, actual quality is increased in case of underestimation of actual bandwidth (left), and actual quality is decreased for overestimation (right).

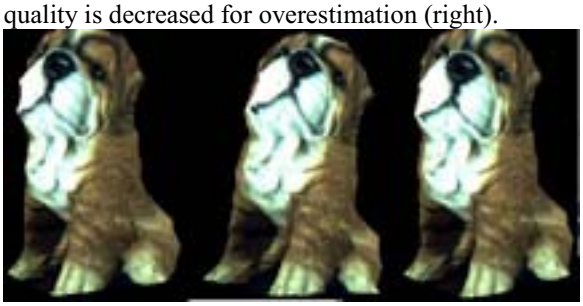

Fig. 10: Initial estimated texture (middle), increased quality (left) and decreased quality (right).

6. Conclusion and future work

The adaptive TexMesh model proposed in this paper applies scale-space analysis to extract feature points at different scales. Quality of a texture fragment is determined by the number of feature points, and the current bandwidth in a given time period. We apply LoG to detect zero crossings at each scale and generate statistics on fragment map during preprocessing. These statistics are then used during runtime for efficient extraction and transmission of texture data. Experimental results show that this adaptive approach utilizes bandwidth more efficiently, and thus provides better control on QoS for online transmission of 3D objects.

In the current implementation, we use the Intel JPEG Library. We plan to combine the advantages of both the wavelet and TexMesh approaches to develop a better coding technique for transmitting textured meshes. In future work, we will perform experiments with more 3D models and formulate the relation between viewing distance, scale, and the quality tolerance interval ΔQ. We will also look into the effect of other factors on perceptual quality and compression.

7. References

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

[2] L. Balmelli,G. Taubin and F. Bernardini, "Space-optimized texture maps," Eurographics, 2002.

[3] P. Boulanger, O. Jokinen and A. Beraldin, "Intrinsic filtering of range images using a physically based noise model," VI 2002, Calgary, Canada.

[4] D. Brodsky and B. Watson, "Model simplification through refinement", Proc. of Graphics Interface 2000.

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

[6] I. Cheng, A. Basu, Y. Zhang and S. Tripathi, "QoS Specification and Adaptive Bandwidth Monitoring for Multimedia delivery," Proc. IEEE EUROCON, 2001.

[7] I. Cheng and P. Boulanger, "Scale-Space 3D TexMesh Simplification," IEEE Int'l Conference on Multimedia, 04.

[8] D. Cohen-Or, Y. Mann, S. Fleishman, "Deep Compression for Streaming Texture Intensive Animations", Siggraph, August 8-13, 1999.

[9] J. Cohen, A. Varshney, D. Manocha, G. Turk and H. Weber, "Simplification Envelopes", Proc. Siggraph 1996.
[10] J. Cohen, M. Olano and D. Manocha, "Appearance-Preserving Simplification", Siggraph'98.
[11] M. Garland and P. Heckbert, "Surface Simplification sing quadric error metrics", Siggraph'97 p209-216.
[12] M. Garland and P. Heckbert, "Simplifying Surfaces with Color and Texture using Quadric Error Metrics", IEEE Visualization 98.
[13] H. Hoppe, "Progressive meshes" Proceedings of SIGGRAPH 1996, L.A. pp. 99-108
[14] A. Kuijper and L. Florack, "Logical filtering in scale space", Institute of Information and Computing Sciences, Utrecht University, Technical Report, 2001.
[15] T. Lindberg, "A scale selection principle for estimating image deformations," ICCV, Cambridge, MA, 1995.
[16] P. Lindstrom *et al*., "Level of Detail Management for Real-Time Rendering of Phototextured Terrain", Technical Report TR-95-06, Graphics, GeorgiaTech, Atlanta, GA.
[17] P. Lindstrom and G. Turk, "Image-driven simplification", ACM Transaction On Graphics, 2000.
[18] D. Luebke, "A Developer's Survey of Polygonal Simplification Algorithms", IEEE CGA, May/June 2001.
[19] M. Okuda and T. Chen, "Joint Geometry/Texture Progressive Coding of 3D Models", IEEE Int'l Conf. On Image Processing, Vancouver, Sep. 2000.
[20] Y. Pan, I. Cheng, and A. Basu, "Quantitative metric for estimating perceptual quality of 3D objects," IEEE ICIP, Barcelona, 2003.
[21] E. Shaffer and M. Garland, "Efficient Adaptive Simplification of Massive Meshes", IEEE Visualization 01.
[22] M. Soucy, G. Godin and M. Rioux, "A texture-mapping approach for the compression of colored 3D triangulations", The Visual Computer (1996) 12: 503-514.
[23] P. Sander, J. Snyder, S. Gortler and H. Hoppe, "Texture mapping progressive meshes", Siggraph 2001.
[24] G Turk, "Generating texture on arbitrary surfaces using reaction-diffusion", Siggraph, July 1991, p289-298.
[25] G Turk, "Re-tiling polygonal surfaces", Siggraph, 92.
[26] A. Witkin, "Scale-space filtering," International Joint Conference on AI, 1983, pp. 1019-1022.
[27] J. Xia, J. El-Sana and A. Varshney, "Adaptive Real-Time Level-of-detail-based Rendering for Polygonal Models" IEEE Trans. on Visualization and CG, June 1997.
[28] Y. Yu, I. Cheng and A. Basu, "Optimal adaptive bandwidth monitoring," IEEE Trans. on Multimedia, September, 2003.
[29] Y. Yu, A. Ferencz and J. Malik, "Compressing Texture Maps for Large Real Environments", Siggraph'00 Sketch.

Appendix A - Proof of Theorm 1:

The time deviation ΔT_k caused by the k^{th} fragment can be expressed as $\Lambda + \varepsilon_k$ where Λ is the average deviation. Let ς_k be $\ln(n-k+1)$, as defined in Algorithm 1.

After 1^{st} fragment is transmitted, ΔT_1 is allocated to the remaining n-1 fragments as follows:

$\Delta T_1 = \Delta T_1 * (1/(2*\varsigma_1) + 1/(3*\varsigma_1) + \ldots + 1/(n*\varsigma_1))$

After 2^{nd} fragment, ΔT_2 is allocated to the remaining n-2 fragments as follows:

$\Delta T_2 = \Delta T_2 * (1/(2*\varsigma_2) + 1/(3*\varsigma_2) + \ldots + 1/((n-2)*\varsigma_2) + 1/((n-1)*\varsigma_2))$

…..

In the last two allocations,

$\Delta T_{n-2} = \Delta T_{n-2} * (1/(2*\varsigma_{n-2}) + 1/(3*\varsigma_{n-2}))$; allocated to $(n-1)^{th}$ and n^{th} fragments,

$\Delta T_{n-1} = \Delta T_{n-1} * (1/(2*\varsigma_{n-1}) + 1/(2*\varsigma_{n-1}))$; allocated to n^{th} fragment.

Since there is no other fragment after n, the n^{th} fragment has to share 100% of ΔT_{n-1}. est_n is revised by adding the cumulated compensating time Δt_n applied to the n^{th} fragment, and we compute Π as follows:

$\Pi = est_n + \Delta t_n - act_n$

$\Pi = d_n / \beta_0 + (\Delta T_1/(n*\varsigma_1) + \Delta T_2/((n-1)*\varsigma_2) + \ldots + \Delta T_{n-2}/(3*\varsigma_{n-2}) + \Delta T_{n-1}) - d_n / \beta_n$

$= (d_n / \beta_0 - d_n/\beta_n) + (\Delta T_1/(n*\varsigma_1) + \Delta T_2/((n-1)*\varsigma_2) + \ldots + \Delta T_{n-2}/(3*\varsigma_{n-2}) + \Delta T_{n-1}/(2*\varsigma_{n-1})) + (\Delta T_{n-1}/2)$

We define $\Pi = E_{est} + E_{comp} + E_{alloc}$

E_{est} – Estimation Error $\Delta T_n = (d_n / \beta_0 - d_n/\beta_n)$ caused by the discrepancy between the historic average and the actual bandwidth for n^{th} fragment. Note that the 1^{st} d_n is before adjustment of Δt_n, and the 2^{nd} is after.

E_{comp} – Compensation Error $\Delta t_n = (\Delta T_1/(n*\varsigma_1) + \Delta T_2/((n-1)*\varsigma_2) + \ldots + \Delta T_{n-2}/(3*\varsigma_{n-2}) + \Delta T_{n-1}/(2*\varsigma_{n-1}))$ allocated from fragments 1 to n-1, shared by the n^{th} fragment.

E_{alloc} – Allocation Error $(\Delta T_{n-1}/2) =$

$(d_{n-1} / \beta_0 - d_{n-1}/\beta_{n-1})/2$, incapable of allocating further.

E_{comp} can be further analyzed by splitting ΔT_i into ε_i and Λ:

$E_{comp} = (\varepsilon_1/(n*\varsigma_1) + \varepsilon_2/((n-1)*\varsigma_2) + \ldots + \varepsilon_{n-2}/(3*\varsigma_{n-2}) + \varepsilon_{n-1}/(2*\varsigma_{n-1})) + \Lambda*(1/(n*\varsigma_1) + 1/((n-1)*\varsigma_2) + \ldots + 1/(3*\varsigma_{n-2}) + 1/(2*\varsigma_{n-1}))$

$\le (\varepsilon_1/(2*\varsigma_{n-1}) + \varepsilon_2/(2*\varsigma_{n-1}) + \ldots + \varepsilon_{n-2}/(2*\varsigma_{n-1}) + \varepsilon_{n-1}/(2*\varsigma_{n-1})) + \Lambda*(1/(n*\varsigma_1) + 1/((n-1)*\varsigma_2) + \ldots + 1/(3*\varsigma_{n-2}) + 1/(2*\varsigma_{n-1}))$

$= \Lambda(1/(n*\varsigma_1) + 1/((n-1)*\varsigma_2) + \ldots + 1/(3*\varsigma_{n-2}) + 1/(2*\varsigma_{n-1}))$

because $\sum_{i=1}^{n-1} \varepsilon_i = 0$;

Note that: $E_{comp} \ge (\varepsilon_1/(n*\varsigma_n) + \varepsilon_2/(n*\varsigma_n) + \ldots + \varepsilon_{n-2}/(n*\varsigma_n) + \varepsilon_{n-1}/(n*\varsigma_n)) + \Lambda*(1/(n*\varsigma_1) + 1/((n-1)*\varsigma_2) + \ldots + 1/(3*\varsigma_{n-2}) + 1/(2*\varsigma_{n-1}))$

So we establish: $E_{comp} = \Lambda(1/(n*\ln(n)) + 1/((n-1)*\ln(n-1)) + \ldots + 1/(3*\ln 3) + 1/(2*\ln 2))$

Since $\frac{1}{x\ln(x)}$ is a continuous decreasing function, the sum can be bounded using integration:

$$\int_3^{n+1} \frac{1}{x\ln(x)}dx \le \sum_{i=3}^{n} \frac{1}{i\ln(i)} \le \int_2^{n} \frac{1}{x\ln(x)}dx$$

Thus if $\Lambda \ge 0$, $(\ln|\ln(n+1)| - \ln|\ln(3)| + 1/2\ln(2))\Lambda \le E_{comp} \le (\ln|\ln(n)| - \ln|\ln(2)| + 1/(2\ln(2)))\Lambda$

and if $\Lambda < 0$: $(\ln|\ln(n+1)| - \ln|\ln(3)| + 1/2\ln(2))\Lambda \ge E_{comp} \ge (\ln|\ln(n)| - \ln|\ln(2)| + 1/(2\ln(2)))\Lambda$

Therefore, we have proved the upper and lower bound of Π in Theorem 1.