Efficient 3D Object Simplification and Fragmented Texture Scaling for Online Visualization

Irene Cheng
Department of Computing Science, Univ. of Alberta, Edmonton, CANADA, [lin]@cs.ualberta.ca

Abstract

Visualization of 3D images is becoming more commonplace for a variety of applications including online games and e-commerce. For efficient online visualization of 3D objects it is necessary to quickly adapt 3D models (including the wireframe and texture) to the available computational or network resources. In this paper we propose enhancements to 3D model simplification based on simplification envelopes (ESE) and curvature variations (CPD) on the surface of an object. We compare the two approaches and demonstrate that the CPD method is much faster than ESE while producing comparable results. A technique for texture simplification driven by model simplification is also proposed. Experimental results comparing the simplified models and the computational time demonstrate the feasibility of our approach.

1. Introduction

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 wireframe). For example, the Zoomage 3D scanner (Fig. 2, right) can produce texture of 200 mega pixels and wireframes with over 2 million triangles; or the Stanford bunny (Fig. 1, right) commonly used as a test object has 69,000 triangles. The trend in multimedia applications is to use more and more polygons in order to produce photo-realistic 3D scenes. However, a large number of polygons impose another challenge to researchers in terms of storage, processing, rendering and transmission. For example in Fig.1, when the mesh is closer to the viewpoint, more polygons can represent better detail, but when the object is farther away, keeping the same number of polygons is not necessary. In Fig.2, only 180 polygons are rendered without loosing significant details on the object. Rendering time is also saved, by reducing the number of polygons from 1800 to 180. Simplification is also useful for online visualization of 3D objects; for example the bandwidth between a server and a client (viewing workstation) can be accurately monitored [5] and the quality of a model and associated texture can be adjusted to allow the best possible visualization within a given time interval.

Level-of-detail (LOD) [1-3] 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 the 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.

Given a sample of range data generated from laser scanning, a simple way to produce different levels of detail is to take subsets from the sample. Decrease of sample size means decrease in detail. This method is simple but does not represent high-density areas with more scan points when merging polygons. To overcome this problem, we need to apply different simplification strategies to regions of different densities. 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. Our paper introduces an enhanced version of the simplification envelopes algorithm (ESE) and a Critical Point Detection (CPD) technique, both using a strategy list to apply different levels of simplification to different parts of an object. A comparison is made between ESE and CPD.

Fig.1: An example of a 1800 polygon 3D object at different distances (left and middle), and Stanford bunny (right).

Fig.2: (Left) wireframe of same object as in Fig.1 using 180 polygons, texture mapped object (middle), and Zoomage™ 3D scanner from TelePhotogenics (right) used to capture 3D data.
Experimental results show that CPD is much faster while generating similar simplification, thus it was chosen as the preferred method for online visualization. We also outline a method to scale different parts of the texture in accordance to a simplified model and the monitored bandwidth for maintaining the best possible visualization quality for resource constrained online applications.

There are two approaches to handle LOD: static and continuous level-of-detail (CLOD) meshes. In the latter, the mesh topology changes in real-time as the viewpoint moves. In the former, a fixed number of LOD representations is pre-computed and rendered at runtime. When an application does not require gradual changes, the static approach with change of topology at discrete stages is more efficient than the CLOD approach. In the current implementation, ESE and CPD adopt the static approach but they can be enhanced to achieve CLOD based on the viewpoint.

The rest of this paper is organized as follows: Section 2 explains the various simplification approaches. Section 3 discusses the data structure used in this implementation. Section 4 summarizes the experimental results. Section 5 outlines a strategy for texture scaling driven by model simplification. Section 6 gives the conclusion and future work.

2. LOD Simplification Approaches

ESE Computation Model

The original simplification envelope algorithm discussed in [1] applies a user-defined value \( \epsilon \) to each vertex along the vertex normal. Instead of requesting the user to supply the value, ESE computes \( \epsilon \) proportional to the inverse of the distance between the viewpoint and the object center. When the distance increases, a smaller \( \epsilon \) is applied so that fewer intersections occur. In other words, more triangles can be simplified. As a result, objects farther away from the viewpoint will contain less but bigger triangles. This approach is more realistic and convenient than requesting the user to supply an \( \epsilon \). An octree was used in the original simplification envelopes algorithm because of its simplicity. But the octree data structure deals only with the bounding boxes of the elements stored. It is inaccurate when detecting intersections. In this project, a more accurate space-partitioning method, Binary Space Partition (BSP), is used.

Suppose \( \text{Dist}_{vc} \) is the distance between the viewpoint and the object center, the modification factor \( \epsilon \) along the vertex normal is defined by: \( \epsilon \propto (1/\text{Dist}_{vc}) \)

If \( n \) is the maximum number of recursion applied to a vertex and \( v \) is the unit vector representing the vertex normal, at each loop level \( L_i \), a vertex is shifted by: \( v(\epsilon/n) \)

Thus the maximum movement of a vertex is constrained to \( \epsilon \). A triangle stops to shift when an intersection occurs, or when the maximum number of loops \( n \) is reached. After the vertices are processed recursively, each vertex is assigned a LOD value \( L_i \), i.e., \( 1 \leq L_i \leq n \), denoting the loop level at which an intersection occurs. If no intersection occurs after \( n \) loops, \( L_{n+1} \) is assigned.

CPD Computation Model

Instead of detecting intersections, CPD detects the angle between the normals of adjacent triangles. The deviation is defined by a value \( \cos \theta \), \( 0 \leq \cos \theta \leq 1 \), which is computed by taking the dot product of the two normalized vectors \( v_1 \) and \( v_2 \).

\[ V_2 - V_1 \]

**Fig.3**: Vertex normal (solid arrow) and the normals (dotted arrow) of two adjacent triangles.

When face1 and face2 are nearly parallel, \( \cos \theta \) is close to 1. In other words, a vertex with a high \( \cos \theta \) value can be removed while one with a low \( \cos \theta \) value should be maintained in order to preserve details. Whether or not a vertex should be removed is controlled by a value \( T \). A vertex is removed if its associated \( \cos \theta \) value is greater than \( T \). Suppose \( \text{Dist}_{vc} \) is the distance between the viewpoint and the object center, the value \( T \) is defined by:

\[
T = (1- \text{Dist}_{vc}/\text{Dist}_{\text{max}})
\]

\( \text{Dist}_{\text{max}} \) is the maximum distance possible between the viewpoint and the object center. As the object moves away from the viewpoint, \( T \) will become smaller. More vertices will be removed from the mesh which will contain lesser triangles. This strategy is based on the observation that human perception is less sensitive to detail at a farther distance.

Instead of assigning a single value \( T \), \( n \) different \( T \)s in increasing order can be used in \( n \) loops. Similar to ESE, after the vertices are processed recursively, each vertex is assigned a LOD value \( L_i \), i.e., \( 1 \leq L_i \leq n \), denoting the loop level at which \( \cos \theta > T \). If the condition is not satisfied after \( n \) loops, \( L_{n+1} \) is assigned. Note that extending curvature based simplification with varying \( T \) relating to the distance of an object and associating it to varying simplification levels allows an intuitive integration with online applications, where bandwidth is monitored based on which a simplification level is selected resulting in effectively adjusting the distance and quality of a 3D object visualized at a client site.

Simplification Strategy

For ESE and CPD, there are two approaches to determine whether a vertex should be removed:

The first approach defines a simplification constraint \( S \). The LOD value \( L_i \) at each vertex is compared with \( S \). If \( S \leq L_i \), the triangles are simplified, otherwise the triangles are not simplified. The smaller the value of \( S \), the less original triangles will be retained. This will result in an integrated mesh with two levels of detail corresponding to higher and lower density regions respectively.

In contrast to the above, the second approach allows a maximum of \( n+1 \) levels of detail. This is achieved by associating different simplification functions \( F_i \) with each LOD value \( L_i \). Different strategies can be used by different \( F_i \). For
example, four triangles can be merged into two in $F_1$, eight triangles can be merged into two in $F_2$, and so on. A combination of strategies enables smaller triangles to be retained in regions where detail needs to be preserved, and bigger triangles put into regions where less detail is required.

Fig.4: A sample of 3D points (left), texture (right).

3. Data Structure

A 3D mesh is constructed by triangulation using a set of sample points in three-dimensional space (Fig.4, left). Sample points are generated from the Zoomage3D laser scanner. The data structure used in this project is designed for storing the original high resolution scan points, from which lower resolution versions can be sub-sampled.

The data structure is composed of three main components: the grid, the mesh and the strategy (or function) list.

The Mesh $M$

The mesh structure $M$ contains a list of triangles. Instead of storing the actual coordinates, each triangle is defined by three gridIndices. These triangles are arranged in strips. Stripification [4] is a technique to minimize the number of vertices transmitted avoiding duplications. The $M$ structure is dynamically changed in each loop. It stores the modified version of the mesh computed based on the value $\varepsilon$ (ESE) or dot product (CPD), and the max number of loops specified. The structure $M$ references the $G$ structure through gridIndices.

The Grid $G$

While the content of the $M$ structure is dynamic, the range data stored in the $G$ structure is static. The scan points from the original high-resolution sample are stored in an $x$ by $y$ grid representation, where $x$ is the number of scan points horizontally and $y$ is the number of scan lines vertically. Re-meshing from the original sample set is therefore possible.

Each grid element $G_{ij}$ contains the following data members:

- $gridIndex$ identifies an element $G_{ij}$ in the grid.
- $pixel$ defines a vertex in 3-dimensional space.
- $texel$ defines the 2d coordinates in the texture map corresponding to the vertex.

- $intersect$ has default value of false until intersection occurs by shifting a triangle along vertex normal or dot product $>$ $T$.
- $removed$ has a default value of false unless the vertex is removed during simplification.
- $LOD$ denotes the loop number at which an intersection first occurs. It is also an index used to reference a particular simplification function in the strategy list.

The Strategy List $F$

The strategy structure $F$ contains a list of simplification strategies, which can be applied to different regions of the mesh. For example, $F_i$ is applied to the neighborhood where the vertices have been assigned a LOD value of $L_i$. The idea is that if $L_i$ is less than the strategy constraint, the associated vertex will not be simplified. The strategy $F_i$ can also store information on how triangles should be merged.

4. Experimental Results

Java3D is used in the current implementation. The experiments were performed on a Pentium III laptop running Windows Millennium. The dog object shown has a height of 6 inches.

Fig.5: ESE: (left) intersecting faces; (middle) simplified non-intersecting faces, texture-mapped; (right) mesh combining non-simplified and simplified faces.

<table>
<thead>
<tr>
<th># of faces original</th>
<th># of faces final</th>
<th>Time (5 loops, $\varepsilon$=-0.01 / loop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>648</td>
<td>520</td>
<td>1</td>
</tr>
<tr>
<td>1620</td>
<td>1616</td>
<td>5</td>
</tr>
<tr>
<td>3240</td>
<td>3233</td>
<td>39</td>
</tr>
<tr>
<td>6480</td>
<td>6480</td>
<td>194</td>
</tr>
</tbody>
</table>

An important observation about ESE is that detecting intersection is expensive, especially when using a high-resolution mesh. It is also noted that a reasonable simplification is only obtained for meshes in the range of a few hundred to a thousand vertices, for the given example. This is because when a mesh is of very low or very high density, the angles between faces tend to be small and intersections can easily occur.

In the experiments, both ESE and CPD start from an original mesh of 648 triangles. We can see that the detected regions of higher detail are similar, but CPD is more efficient in generating the results. Also, ESE does not seem to be able to simplify high-density meshes very well.
Fig. 6: CPD: (left) non-simplified faces; (middle) simplified faces, texture-mapped; (right) texture-mapped model combining non-simplified and simplified faces.

<table>
<thead>
<tr>
<th>CPD</th>
<th># of faces original</th>
<th>648</th>
<th>1620</th>
<th>3240</th>
<th>6480</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cos θ=0.4: time (sec.)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td># of faces final</td>
<td>384</td>
<td>838</td>
<td>1672</td>
<td>3600</td>
<td></td>
</tr>
<tr>
<td>Cos θ=0.8: time (sec.)</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td># of faces final</td>
<td>505</td>
<td>1101</td>
<td>2668</td>
<td>5659</td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that ESE has O(n²) complexity whereas CPD has O(n) complexity where n is the number of vertices, thus CPD is preferable for large datasets and real time online applications.

5. Fragmented Texture Scaling
The discussion thus far concentrated on adaptive wireframes and using a strategy list to select the appropriate LOD. However, for online applications a majority of the bandwidth is used for texture transmission. An easy way to simplify texture is to scale the resolution of different parts of the texture depending on the distance from the viewer or the amount of detail locally in the model transmitted. The total bandwidth requirement can be estimated as follows:

Let TM(i), i = 1, …, K be the estimated compressed size of a texture fragment given compression quality level i. For example, we can consider the quality parameter in JPEG compression or adjust the resolution of the texture to varying levels.

Let N_L(F_j) denote the number of faces at a simplification level L corresponding to a texture fragment F_j. A quality function, Q(N_L(F_j)), is defined that relates the number of faces in a fragment to the texture compression quality of the fragment; Q is a non-decreasing function which maps from 0,1,2,… to the values 1, …, K.

Let M be the total number of texture fragments and B be the estimated bandwidth for a given time interval. We need to have:

$$\sum_{i=1}^{M} TM(Q(N_L(F_j))) \leq B \quad (I)$$

The level L of simplification can be chosen to maximize the value on the left hand side of (I). It is assumed that the cost of wireframe transmission is negligible compared to color texture transmission. The model proposed here is simple but fast and easy to estimate in real time online applications. A more accurate implementation would store the actual compressed sizes of all fragments at different quality levels in a 2D table and use these values instead of estimates.

An example of the texture scaling process is shown below. The original texture is divided into predefined fragments each of size 1/6th the original image. Depending on the wireframe detail the texture quality can be adjusted. Fig. 7 shows the top half of the dog texture with the middle third having higher quality than the two sides, which are noticeably blocky. The total size of two high quality and four low quality fragments for the dog was 153 Kbytes compared to 330 Kbytes for high quality throughout.

Fig. 7: Top half of transmitted dog texture.

6. Conclusion and Future Work
In this paper we proposed using enhanced 3D model simplification algorithms and monitored bandwidth to adapt the quality of different texture fragments for efficient online visualization. The issue of optimally determining the number and size of fragments needs to be considered in future work. We need to conduct further research on “perceptual quality measurement” of the simplified objects using various approaches to obtain a meaningful comparison of the quality of the transmitted 3D objects.

We also plan to integrate the simplification software to our past research on optimal bandwidth monitoring and develop strategies for efficient implementation over wireless networks.

7. References