Joint Geometry/Texture Progressive Coding of 3D Models

Masahiro Okuda and Tsuhan Chen

Dept. of Electrical and Computer Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213, USA
Email: masa@andrew.cmu.edu, tsuhan@ece.cmu.edu

Abstract

Files of 3D models are often large and time-consuming to download. Most 3D viewers need the entire file to display a 3D model even when the user is interested only in a part of or a low-resolution version of the model. Therefore, progressive coding of 3D models is desired. However, existing work studies either the progressive coding of the geometry only, or progressive coding of the texture only. In this paper, we propose a joint geometry/texture coding technique for 3D objects. Both of the geometry and the texture are progressively coded and transmitted to the viewer. The most perceptually important bits are sent before the less important bits, allowing the users to stop the download at any time and yet retain the best quality available at that time.

1. Summary

Due to the rapid development of computer and information technology, 3D modeling capabilities are becoming an increasingly important part of many fields, including video games, CAD/CAM, medical imaging, and so on. However, files of 3D models are often large and time-consuming to download. Moreover most 3D viewers need the entire file to display the model even when the user is interested only in a part of or a low-resolution version of the model. Since the computers used to view the file range from low-end PCs to top-of-the-line supercomputers, and since the files are sent over slow telephone line as well as faster ISDN, the level of detail actually required may vary considerably. The progressive representation of the 3D models is desired to satisfy the variety of demands.

There exist some works on 3D model coding such as a compressed binary format (a list of current Web3D working groups can be found at [1]). The MPEG-4 Synthetic/Natural Hybrid Coding subgroup (MPEG-4 SNHC) is also working in similar areas [2]. In [3], [4] and [5], progressive 3D coders have been proposed. In our previous work [6], we proposed progressive 3D coding using the vertex decimation scheme, which provides fully progressive bitstreams.

However, most existing work considers only progressive coding of geometry of the 3D models. The 3D models of the VRML usually have one or more attributed data such as normals, colors, textures and so on. If the 3D models need to look authentic, texture maps are required. In this paper, we propose a joint geometry/texture coding scheme for the 3D models, resulting in progressive bitstreams. In our codec, the most perceptually important bits to represent the 3D models are sent before the less important bits, allowing the users to stop the download at any time.

2. Joint Geometry/Texture Coding

2.1. Vertex decimation scheme

The proposed method is an extension of our previous work in [6]. The encoder removes vertices until we are left with a base mesh which has only a small fraction of the vertices and triangles in the original model. To send a 3D model progressively, we start by sending the base mesh, which is simply the vertex positions and triangle vertices. From this base mesh, enhancements to the model are sent, vertex by vertex, until all the vertices are restored.

The process of vertex decimation is as follows. In a triangular mesh, there is a ring of triangles that surround every vertex. For example, see Fig. 1. Vertices that satisfy either one of the following two conditions are not decimated by the algorithm so as not to destroy the topology and the appearance of the model.

1. The vertex has surrounding triangles that do not form a closed ring.
2. The vertex has extraneous triangles connected to it.

To ensure that the most perceptually important vertices are sent before the less important vertices, we must have a way of measuring the importance of vertices. Two measures of the importance are introduced. The measure $v(i)$ we implemented is to estimate the difference in the volume caused by the decimation, by forming tetrahedrons with the removed vertex as the apex and the new triangles as the base, and adding up the volumes of these tetrahedrons. The measure $v(i)$ is similar to the "curvature" of the mesh at the vertex. The other measure $c(i)$ is texture similarity. During the simplification procedure, some triangles are decimated.

Submitted to IEEE International Conference on Image Processing 2000 in Vancouver

In case that each triangle has a corresponding small texture, some textures are lost by the simplification. Thus, in order to preserve the appearance of the models, we need to carefully choose the vertices being decimated. In this paper, we adopt the color difference as the criterion. The idea of this second measure is as follows.

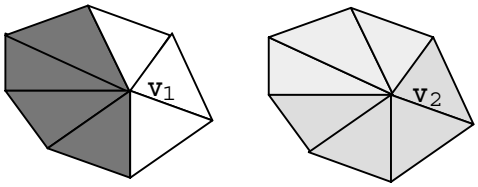


Fig. 1: Illustration of 3D surfaces:
(a) triangles with different colors
(b) triangles with similar colors

Consider two vertices on 3D surfaces, v1 and v2 in Fig. 1. While the triangles connected to v1 have different color information, those of v2 have similar colors. In order to preserve the appearance of the models, v2 should be decimated prior to v1 even when these two have a same curvature. This second criterion seeks to find the importance of vertices in terms of the color difference. In practice, we find the average of each component, and then, we use the sum of the absolute differences between the averages as the measure. In our framework, we calculate the weighted sum of the two measures for each vertex as in (1) and then decide which vertex should be decimated.

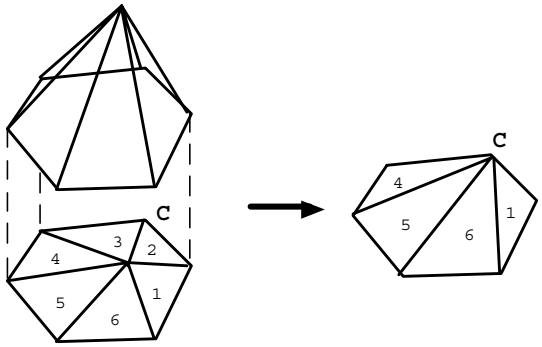
$$m(i) = a_0 v(i) + a_1 c(i) \qquad (1)$$

Vertices with large $m(i)$ are considered more perceptually important. Therefore, they are decimated later in the encoding process, and sent earlier during transmission. Note that user can freely select the weights, a_0 and a_1, depending on the user's preference on the importance of geometry and color information.

2.2. Re-triangulation and texture re-mapping

Once a vertex is decimated, re-triangulation and texture re-mapping are needed to fill up the hole caused by the decimation. For this, we need to consider two ways of corresponding the 3D geometry with the texture map [7]. In one case, each vertex in the 3D model corresponds to one point in the texture map. In the other case, each triangle in the 3D model corresponds to a triangle in the texture map. In case that each vertex has one texture correspondence, once a vertex is decimated, the textures which belong to the triangles originally connected to the vertex can be re-mapped directly to the base left by the decimation. Since the texture coordinates of the remaining vertices will remain the same, any re-triangulation scheme produces similar results. However in case that each triangle has one texture correspondence, the re-triangulation and the re-mapping significantly affect the decimated model. In the following, we discuss the re-triangulation and the texture re-mapping schemes used in our algorithm for the case of correspondence for each triangle.

First, among all vertices connected to the vertex that is considered, we find the vertex that has the closest distance from the removed vertex. Then, the removed vertex is mapped to the closest vertex, which results in a triangle fan (see Fig. 2). The textures owned by the triangles that are retained are simply re-mapped to the new triangles. For example, in Fig. 2, the decimated vertex is moved to the vertex **C**. The textures of the triangles 1, 4, 5 and 6 are mapped to the new four triangles.



C : closest vertex

Fig. 2: Re-triangulation and texture re-mapping

In the bitstream, each decimation is encoded as a seven-tuple: the index of the closest vertex on the edge, the indices of the two vertices on which we start the fan triangulation, the number of triangles to traverse, the x, y, z coordinates of the decimated vertex, and two texture coordinates, s and t, for each vertex in the two triangles removed by the decimation. Vertices are numbered in the order they are sent in the base mesh, and each new vertex is assigned the next available index.

2.3. Progressive texture coding

As is described above, there are two types of texture correspondence in the VRML format, correspondence for each vertex and for each triangle. In the former case, one texture is mapped on a unit of a model. Since this type of texture is often as smooth as ordinary images, we adopted wavelet coding for texture compression, resulting in SNR progressive bitstreams. Both of the geometry and the texture get finer progressively as more bits are received.

In the latter case, the texture map contains many small and large triangles. See Fig. 4 for an example. For this type of texture, we encode each triangle independently and send it to the decoder. The coding method we employ is as follows.

First consider a square circumscribing the triangle. We choose the size of the square to be a multiple of 8. We fill up the pixels outsize the triangle with the pixels at boundary of the triangle by horizontal/vertical repetitive padding similar to what is used in MPEG4. Fig. 3 shows the padding procedure, in which black and grey pixels outside the triangle are padded by vertical and horizontal repetitive padding, respectively. Then, we compress the square using the DCT, scalar quantization, zigzag scan and Huffman coding, similar

to the JPEG algorithm. In our framework, only the textures required to render a current level of the model are transmitted. Hence, progressive texture coding is achieved.

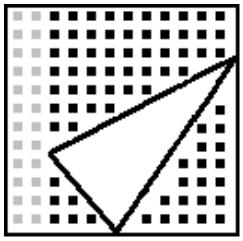


Fig. 3 Horizontal/Vertical Repetitive Padding

3. Experimental Results

We implemented a viewer to progressively display the 3D models coded by our algorithm. Once sufficient bits are downloaded from the server to display more detail, the new model is updated on the screen. While the file is being downloaded, the user can change the viewpoint to examine the model. If the user is uninterested in the model, the downloading can be stopped at any time. Some VRML models, which have one correspondence for each triangle, obtained from [8] were used in the experiments. Fig. 4 shows what a model looks like as it is being loaded through a 56K modem dial-up connection. The model (a), (b) and (c) were decoded using 8%, 28% and 100% of the bitstream, respectively. It can be seen that even a low level version of the model gives the user a very good idea of what the complete model would look like. Without progressive coding, we would need about 86 seconds to download and see the model. Since we use the color difference as the measure of importance for vertices, color patterns (such as an edge between white and black feathers) are kept during the vertex decimation procedure.

Table.1 shows the compression efficiency of the proposed algorithm. Both the original and progressive files are compressed by gzip. Note that all information about the meshes such as vertex coordinates and triangle correspondences are coded losslessly, while the lossy compression scheme in Sec.2.3 is applied to the textures. As can be seen from the table, about 58% of bits are saved compared to the original files, despite the fact that they are coded progressively.

4. Conclusion

In this paper we demonstrated joint geometry/texture progressive coding of 3D models. We have created tools that code VRML files into progressive bitstreams, and a browser that allows the user to download and view these files progressively. We examined the compression efficiency and showed that it is not compromised by the progressive coding process. Our viewer has been implemented and fully tested. It is available for download on the web site: (http://amp.ece.cmu.edu/).

References

[1] Web3D Consortium Working Groups, http://www.web3d.org/fs_workinggroups.htm

[2] MPEG-4 SNHC web page, http://www.es.com/mpeg4-snhc

[3] MPEG-4 SNHC, Gabriel Taubin, editor, "SNHC Verification Model 9.0 [3D Mesh Encoding]", W2301, 7/10/98

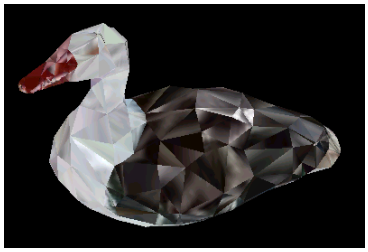
[4] G. Taubin and J. Rossignac, "Geometric Compression through Topological Surgery", ACM Transactions on Graphics, 1998. Also IBM Research TR RC-20340, Jan 1996.

[5] J. Li and J. Kuo, "Progressive Coding of 3-D Graphics Models", Proceedings of the IEEE, Vol. 86, No. 6, June 1998

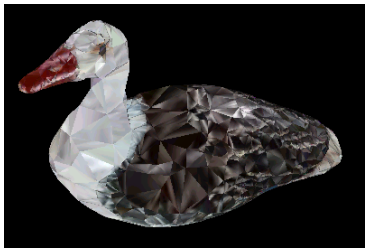
[6] B. Koh and T. Chen, "Progressive VRML Browser", IEEE International Workshop on Multimedia Signal Processing, Sep 1999.

[7] J.D. Foley, "Computer Graphics: Principles and Practice", Addison-Wesley Pub Co.

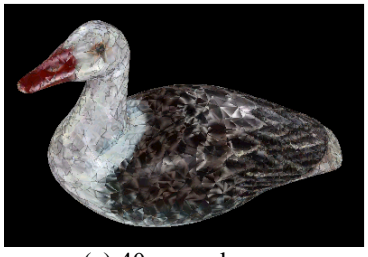
[8] http://www.vit.iit.nrc.ca/3D/Pages_HTML/3D_Models.html



(a) 3 seconds



(b) 11 seconds



(c) 40 seconds

Fig. 4: Progressively decoded 3D model

Table.1: Comparison of file sizes

Model	Progressive	Original
Duck	282kB	605kB
Totem Pole	291kB	683kB
Vase 1	259kB	651kB

Vase 2	280kB	687kB