A Compression Method for 3D Scenes

Radu-Lucian Lupşa

May 27, 2001

Abstract

The article presents a method for compressing the representations of 3D scenes. It is a non-lossy compression, based on a variation of the Huffman algorithm and some ideeas taken from the LZ77 compression method. The method described has been succesfuly implemented and used in a commercial application.

1 Introduction

3D graphics applications need a representation of 3D scenes they work on, for storing or transmitting the information about the objects. A 3D representation must meet the following requirements:

- it must be able to represent the 3D scene with enough accuracy for the application,
- one must be able to convert it fast enough to the representation required by the 3D engine or by other parts of the aplication,
- it must be space-efficient.

There are two basic methods for describing 3D objects:

1. as geometric bodies
2. by voxels

As the second method leads to huge memory requirements, we will use the first one.

In order to describe the bounding surface of a 3D body, two elements must be specified:

1. the shape of that surface

2. the optical properties of the surface

A body surface can be approximated by an union of elements (or patches), each patch being a (finite) fragment of a plane, of a Bezier surface, or of a B-spline surface.

The relevant optical properties of the patches are:

1. the color (or the texture of that surface)
2. the reflection model
3. the refraction model

The last two properties are omitted from the less elaborated representations, as they can be used for rendering by very costly algorithms only (such as ray-tracing).

If a patch has only one color, we can represent its components; if there is a texture, we represent thar texture as a 2D image (using a normal 2D image representation format) and that image is mapped onto the patch.

Having the patches planar (that is, each patch is a polygon, and the body is therefore a polyhedron) simplifies the computations but the edges are far too visible in the rendering. There are rendering methods for smoothing the rendering; the best known are the Gouraud or Phong methods [1] [7]. Any of those requires an approximation for the (real) normal vector in each of the vertices. However, computing the normals from the patches corners only is peculiar because some of the edges between the patches are to be smoothed, and others are real edges. For that reason, some of the 3D representation formats explicitly represent the normal vector of each patch in each corner.

2 Standard 3D formats

Several 3D formats (for instance, `.obj` (for a front end for OpenGL) and `.3ds` (3D Studio) are constructed the following way

First of all, we have a list of 3D points, a list of 3D vectors, a list of 2D points, and a list of 2D images (the latter being represented using a standard 2D format — for instance, *gif*, *jpeg*, *tiff*, and may even be stored in different files). Next we have the description of the facets (patches). Each facet description contains:

- the vertices list

- the list of the normals in each vertex (in case the face is not planar so the rendering should be smoothed)
- the texturing image
- the 2D points on the texturing image, corresponding to the facet vertices

Each of these pieces of information are in fact the index, in the list at the beginning of the file, of the corresponding 3D point, 3D vector, 2D image, or, respectively, 2D point.

3 Compressing the 3D scene

The methods described above still contain a lot of redondancy. Eliminating this redondancy would lead to a better compression.

In the following we will start from a `.obj`-like 3D format and will try to compress it.

The first source of redondancy consists in the fact that a typical application will output the vertices and vectors in approximatively the same order they are used by the facet descriptions. This sugests us to write in the compressed file the list of differences between the succesive vertex indices of each face, and to compress those differences using the Huffman algorithm [4] [5], with some modifications inspired from other compression techniques.

The first change will be to make an adaptative Huffman algorithm. In the original algorithm, the coding table is computed in a first pass over the input file and written into the compressed file; then the information is encoded using that table.

The modified algorithm will start with a fixed encoding table. As it sees the input data, it computes the frequency table. At some predefined moments (for instance, when the number of already-processed symbols is a power of 2), the encoding table is regenerated based onto the frequency table.

The decoder starts with the fixed encoding table. Relying on it, the decoder can read and decode the first symbols, till the first encoding table recomputing. At that time, the decoder will have exactly the same frequency table as the encoder, and therefore it will generate the same encoding table, so it will be able to continue the decoding process.

The second modification concerns the handling of rarely-used symbols. As we saw earlier in this section, the input symbols for the

Huffman compression are the differences between the indices of two succesive points on a facet. These differences, if the indices are 32-bit integers, lay in the interval $-2^{31}+1..2^{31}-1$, but values above a few hundreds are rare. For that matter, statistical data are irrelevant for predicting future occurences of those values. So, we will slightly change the Huffman algorithm the following way: for the Huffman algorithm, we will consider all values outside the interval, let's say, $-127..127$ as being equal. this way, the Huffman part sees 256 distinct symbols, one for each number in the interval $-127..127$ and one for all the other numbers. For the numbers outside the interval $-127..127$ we output the Huffman code of that symbol plus 32 bits representing the actual value.

Sometimes we have a second source of redondancy in the point and vector components. Let's take the sequence of the x coordinates of the points. If there are points grouped in planes orthogonal to the Ox axis, we get repeating values in that sequence. So, instead of coding the actual values, we will code the distance from the last appearance of that value.

4 Conclusions

The method described in the previous section was implemented by the author and is used in a commercial application for sending descriptions of 3D scenes over the Internet. The scenes are output by a CAD-like program and are between 300kB and 5MB in *obj* format. A simple conversion from text to binary reduces the size to one half, and a *zip*-like program reduces it to 1/4..1/5 of the original size. The compression ratio aquired by the program using the method described above is 1/8..1/10.

References

[1] P. Burger, D. Gilles. *Interactive Computer Graphics.* Addison-Wesley Publishing Company, 1990

[2] F. Preparata, M. Shamos. *Computational Geometry.* Springer-Verlag, 1988

[3] R. Gonzales, R. Woods. *Digital Image Processing.* Addison-Wesley Publishing Company, 1993

[4] AL. SPĂTARU. *Teoria transmisiunii informaţiei — Information Transmission Theory.* Editura Tehnică, Bucureşti 1965

[5] X. MARSAULT. *Compression et cryptage de l'information — Information Compression and Encrypting.*

[6] R. LUPŞA. *A Method for Compressing Static Images Using Spline Functions.* Proceedings of the "Tiberiu Popoviciu" Itinerant Seminar of Functional Equations, Approximation and Convexity, Cluj-Napoca, May 21–25, 1996

[7] C. VAN OVERVELD, B. WYVILL *Phong Normal Interpolation Revised.* ACM Transactions on Graphics, oct. 1997, vol. 16, no. 4

[8] HEE CHEOL YUN, BRIAN GUENTER. *Lossless Compression of Computer-Generated Animation Frames* ACM Transactions on Graphics, Oct. 1997, vol. 16, no. 4

[9] M. LOUNSBERY, T. D. DEROSE, J. WARREN. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type.* ACM Transactions of Graphics, Jan. 1997, vol. 16, no. 1