Vol. 4, No. 4: 00–00

A Fast Depth-Buffer-Based Voxelization Algorithm

Evaggelia-Aggeliki Karabassi, Georgios Papaioannou
and Theoharis Theoharis
Department of Informatics, University of Athens

Abstract. This paper presents a fast and easy to implement voxelization algorithm, which is based on the z-buffer. Unlike most existing methods, our approach is suitable both for polygonal and analytical objects. The efficiency of the method is independent of the object complexity and can be accelerated by taking advantage of widely available, low-cost hardware.

1. Introduction

Volume graphics is a domain that has a rather short history, but it is gaining increasing popularity. Voxel-based models are used in a variety of applications, including (but not limited to) medical imaging, fluid dynamics, CSG, terrain modeling, texture generation, and lately even in computer games.

In many cases, the voxelization of non-discrete models is required, to further manipulate them using techniques applicable to voxel-based objects. Voxelization, [Kaufman, Shimony 86], [Cohen et al. 94] the process of approximating a continuous object by a set of voxels, consists of sampling the initial object and assigning a value to each voxel of a three-dimensional raster.

In this paper, we present a simple and fast algorithm, which rapidly produces volume data from any kind of original model to which a zbuffer can be applied; the algorithm can be used as a previewing stage or in applications where accuracy is not critical.

1086-7651/99 $0.50 per page

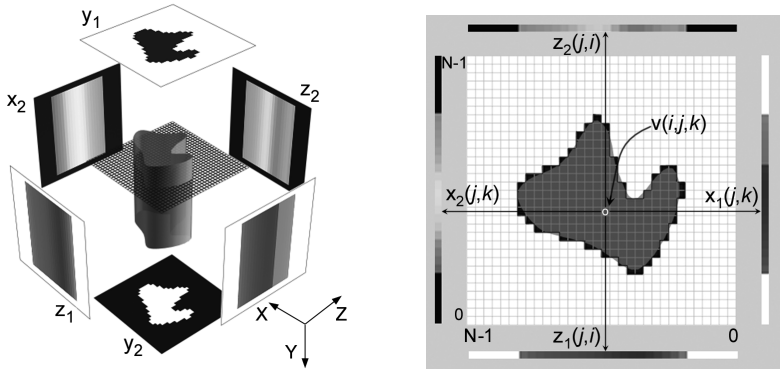

Figure 1. Voxelization of an object using a $32 \times 32 \times 32$ voxel space. (a) Buffer configuration in three-dimensional space. (b) A two-dimensional slice. The grid indicates the voxel space. White voxel = empty; black voxel = object.

2. Algorithm Overview

Our algorithm is based on the creation of volume data using depth information from different views of the object and can be regarded as an application of the z-buffer. Similar work has been conducted [Prakash, Manohar 95] on the voxelization of convex objects using a single pair of depth buffers; however, it is limited to convex objects representing unstructured grid cells and the voxelization process is interwound with the scan conversion step. The algorithm described here can be used for certain non-convex objects and does not require a specific data representation, as we separate the depth buffer calculation from the voxelization. In this manner, we can exploit the hardware z-buffer to increase performance.

Imagine that the object to be voxelized is surrounded by a bounding box, and that each face of the box is a viewing plane. A depth buffer is generated for each face by parallel-projecting the object onto it (see Figure 1). Resolutions for the faces are chosen so that there is a consistent number of pixels along each axis.

For each pair of faces along an axis (e.g. X and $+X$), we obtain for each pixel a minimum and maximum distance for the object. So, any given voxel center will have three pairs of values associated with it, one pair per axis. If a voxels location is inside (i.e., bounded by) all three pairs, then the voxel is inside the object.

The main limitation of the method is that it can miss concavities. If some area of the surface is not visible from any of the six faces, then this area will not be properly voxelized. While this sounds like a serious drawback, a large class of objects can be quickly and successfully voxelized by this algorithm.

If the voxelization is unacceptable, a more general (but slower) solution such as Chen and Fangs [Chen, Fang 98] could be used.

An implementation detail worth noting is that the same viewing matrix can be used for a pair of renderings. This is done by setting the compare logic on one of the renderings to save the z-depth values farthest away instead of closest.

3. Surface Voxelization

We shall now describe how the depth buffer technique can be adapted to voxelize only the object surface. A voxel is part of the object surface if at least one of its coordinates lies close (within a deviation d) to one of the corresponding buffer values, while the other coordinates are bounded by the remaining buffer pairs, as is the case in the main algorithm. The deviation d could be regarded as the surface thickness at each point with regard to the viewing axis. The value of d at each point must be given a value so that (a) the voxel set does not leave connectivity holes and (b) no redundant voxels are created [Kaufman et al. 93].

The surface thickness at each point depends on the surface curvature at the point and on the surface orientation with regard to the viewing angle. A rough estimate of d for a given point with regard to a viewing direction (e.g., $+Z$), which satisfies the above two requirements, can be calculated as the maximum difference between the z value at this point (as stored in the z-buffer) and its four or eight neighbors, depending on the desired connectivity.

As an example, in the two-dimensional case of Figure 2, the surface thickness corresponding to position i of buffer z_1 is equal to 3, ensuring that even surface points "hidden" from z_1 will be voxelized (voxels marked with "O").

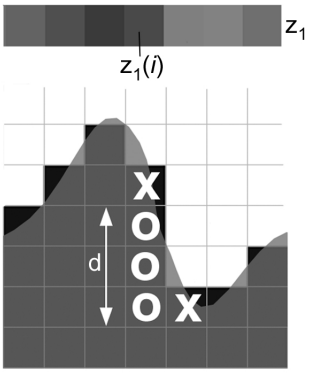

Figure 2. Calculation of the surface thickness d in the two-dimensional case.

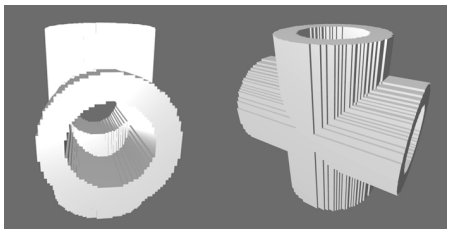

Figure 3. Voxelized model of a hollow cross.

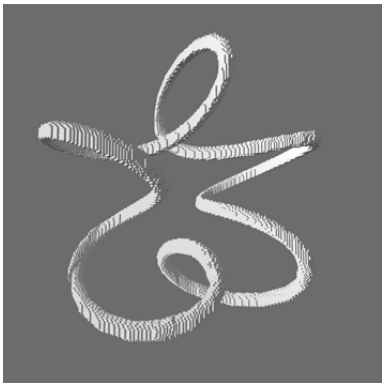

Figure 4. Knot created using surface voxelization

4. Results

All algorithms presented in this paper were implemented in C++, using OpenGL under Windows 98 on a 400 MHz Intel Pentium II system, with a nVIDIA Riva TNT graphics board.

Since our algorithm is based on the z-buffer, its complexity is independent of the object complexity. Of course, the computational time is proportional to the required level of detail, which can be easily controlled by the z-buffer resolution. The computational time is of the order of half a second for 1283 voxelizations and 6 seconds for 2563 voxelizations. Surface voxelization is a bit slower, about 2 and 15 seconds respectively. Figure 3 shows a voxelized hollow cross; Figure 4 shows a knot created using surface voxelization and Figure 5 displays a voxelized truck model. The algorithm can be used to easily produce multiresolutional models, simply by repeated execution for different buffer resolutions, as in Figure 6.

As already mentioned, the algorithm is not suitable for accurately voxelizing objects with hidden cavities (Figure 7). However, we have tested the method with a variety of models of common objects, and approximately 90% produced successful results.

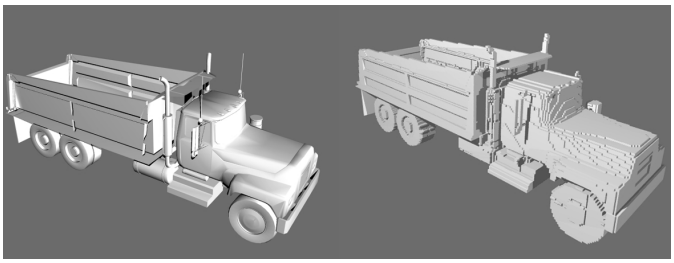

Figure 5. Left: polygonal truck model. Right: Truck voxelized at a resolution of $128 \times 128 \times 128$.

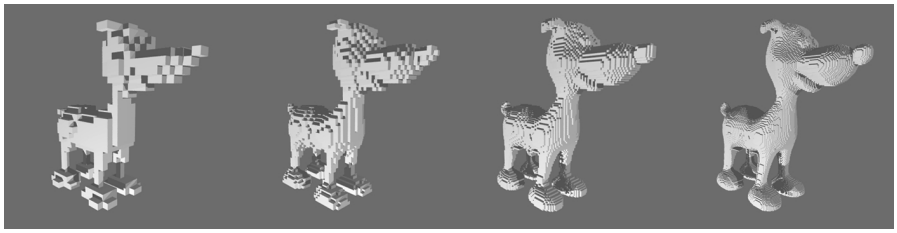

Figure 6. Voxelization of a cartoon dog model using multiple resolution depth buffers. Buffer dimensions from left to right: 32×32, 64×64, 128×128, 256×256.

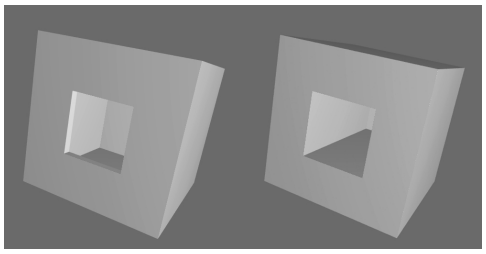

Figure 7. Error in voxelization. Left image shows a hollow cube which is erroneously voxelized as a cube with a hole (right).

References

[Cohen et al. 94] D.Cohen, A.Kaufman and Y.Wang. "Generating a Smooth Voxel-Based Model from an Irregular Polygon Mesh." *The Visual Computer*, 10: 295–305 (1994).

[Chen, Fang 98] H. Chen and S. Fang. "Fast Voxelization of Three-Dimensional Synthetic Objects." *Journal of Graphics Tools*, 3(4): 33–45, 1998.

[Kaufman et al. 93] A.Kaufman, D.Cohen and R.Yagel. "Volume Graphics." *IEEE Computer*, 26: 51–64 (1993).

[Kaufman, Shimony 86] A.Kaufman and E.Shimony. "3D Scan Conversion Algorithms for Voxel-Based Graphics." In *Proc. ACM 1986 Workshop on Interactive 3D Graphics*, pp 45–76. Chapel Hill, NC, 1986.

[Prakash, Manohar 95] C.E. Prakash and S.Manohar. "Volume Rendering of Unstructured Grids–A Voxelization Approach." *Computer Graphics*, 19(5): 711–726 (1995).

Web Information:
Additional images of voxelized objects can be found at:

http://www.acm.org/jgt/papers/KarabassiEtAl00/
Source code is available by contacting the authors.

Evaggelia-Aggeliki Karabassi, Department of Informatics, University of Athens, Panepistimiopolis, Ilisia, Athens 15784, Greece (aggeliki@di.uoa.gr)

Georgios Papaioannou, Department of Informatics, University of Athens, Panepistimiopolis, Ilisia, Athens 15784, Greece ( )

Theoharis Theoharis, Department of Informatics, University of Athens, Panepistimiopolis, Ilisia, Athens 15784, Greece ( )

Received July 16, 1999; accepted in revised form April 18, 2000.