

A Depth Buffer Based Voxelization Algorithm

Aggeliki Karabassi, George Papaioannou and Theoharis Theoharis

Department of Informatics, University of Athens
TYPA Buildings, Panepistimiopolis, Athens 15784, Greece

Contact information :

Aggeliki Karabassi

Department of Informatics, University of Athens

TYPA Buildings, Panepistimiopolis, Athens 15784, Greece

e-mail : aggeliki@di.uoa.gr

Abstract. The term voxelization describes the conversion of an object of any type into volume data, stored in a three dimensional array of voxels. 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 one of the latest developments in computer graphics, yet it is evident that it has the potential to become one of the most useful techniques in three dimensional object manipulation and representation. Voxel-based models [Kaufman, Cohen, Yagel 93] are used in a variety of applications, including (but not limited to) medical imaging [Pommert et al. 92], [Stytz, Frieder, Frieder 91], fluid dynamics, CSG, terrain modeling [Cohen-Or 97], texture generation [Kajiya, Kay 89], [Neyret 98] and lately even in computer games.

In many cases, the voxelization of non-discrete models is required, in order to further manipulate them using techniques applicable to voxel-based objects. Voxelization, that is 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.

The main idea in most voxelization algorithms is to examine whether each voxel belongs to the object or not and assign to the voxel a value of 1 or 0 respectively. This is accomplished by either examining if a voxel's centre lies inside the object, or by selecting all the voxels that are intersected by the object [Kaufman, Shimony 86]. More sophisticated algorithms generate smoother and alias-free models and involve filtering of the volume [Wang, Kaufman 93], [Wang, Kaufman 94], subdivision of the original object [Cohen, Kaufman, Wang 94] or calculation of the exact distance of a voxel from the object surface [Jones 96]. We should mention at this point that the majority of voxelization techniques are oriented to a single type of object, e.g. polygonal meshes, lines, parametric surfaces etc.

In this paper we present a simple and fast algorithm, which produces volume data from any kind of original model to which a z-buffer can be applied. Section 2 describes the basic idea of the method, while in Section 3 we comment on the advantages of the algorithm and compare it to previous methods. Section 4 presents two variations of the method, which voxelize only the contour or produce volume data of varying density. In Section 5 we present some test results.

2. Algorithm Overview

Our algorithm is based on the creation of volume data using depth information from different views of the object and could be regarded, as already mentioned, as an application of the z-buffer. The object is surrounded by three orthogonal pairs of z-buffers, each pair holding depth information for the object from a different viewing angle. Voxelization is accomplished by scanning the volume space and checking whether each voxel is between the limits imposed by the buffers.

Previous work has been conducted in [Prakash, Manohar 95] on the voxelization of convex objects using a single pair of depth buffers. Discussion of the method is limited to convex objects representing unstructured grid cells and the voxelization process is interwound with the scan conversion step. Our algorithm can be used for non-convex objects and does not require a specific data representation, as we separate the depth buffer acquisition from the voxelization. This way we can exploit hardware z-buffer to increase the performance.

2.1. Setting up the buffers

The first step of the method is to place the object in the scene, as we would normally do for rendering, independently of the object type. In order to proceed with the voxelization we need to set up three pairs of depth buffers ($[x_1, x_2]$, $[y_1, y_2]$, $[z_1, z_2]$), facing along the three axes X , Y , Z which define the global coordinate system. Each pair consists of two buffers, perpendicular to the same axis but in opposite sides of the object, in such a way that they are facing each other (Fig. 1).

Buffers of the same pair hold opposite views of the object and correspond to the same viewing axis. The first buffer of each pair x_1 , y_1 , z_1 (looking at R, U, F in Fig. 1) holds the closest depth to the viewer, while the second buffer x_2 , y_2 , z_2 (looking at L, D, B in Fig. 1) stores the maximum distance from the viewer, along the corresponding axis.

Since depth information is usually stored in the z-buffer, that is in a buffer along the Z axis, in our case the object needs to be rotated twice, in order to obtain the “x-buffer” and “y-buffer” pairs. No additional transformations are required to obtain the second buffer of each pair, since this can be easily accomplished by modifying the comparison function used by the z-buffer, to store the greatest distance instead of the smallest. Control over the comparison function is provided by most common APIs, like OpenGL.

In Fig. 1 a greater value (lighter colour) indicates a more distant point, while a smaller value (darker colour) indicates a point closer to the viewer. Points in the background are interpreted as far points in the first

group of buffers and as close points in the second group, since the comparison criterion is reversed.

2.2. Voxelization

Once all six buffers have been created, we may proceed with the actual voxelization step. Let $V = \{ v(i,j,k) : i,j,k \in [0,N) \}$ be the voxel cube. The final voxel-based model detail is imposed by the buffer resolution, so we must choose $N \times N$ sized buffers for a N^3 voxel cube. The voxel cube orientation is defined by the three global axes X, Y, Z .

Conversion to volume data, consists of a triple loop, to cover the voxel space. A voxel $v(i,j,k)$ is related to six projected points on the depth buffers, whose coordinates are (j,k) , (k,i) and (j,i) on the X, Y , and Z pair respectively. The corresponding depths are given by the appropriate buffers: $x_1(j,k)$, $x_2(j,k)$, $y_1(k,i)$, $y_2(k,i)$, $z_1(j,i)$, $z_2(j,i)$ (Fig. 2a).

Voxelization, i.e. deciding whether a voxel belongs to the object (interior or contour) or not, is equivalent to examining if the voxel lies within the boundaries defined by the corresponding depth values of the buffers (Fig. 2b). Since voxel coordinates are defined relative to the voxel cube size (in the interval $[0,N)$), while z-buffer values lie in an implementation dependant range $[depth_{min}, depth_{max}]$, some conversion is necessary for the comparison.

The position of $v(i,j,k)$ if converted to the z-buffer range is :

$$(x, y, z) = (i, j, k) \cdot \frac{depth_{max} - depth_{min}}{N - 1} \quad (1)$$

and $v(i,j,k)$ lies inside the object if and only if :

$$\begin{aligned} & x_1(j,k) \leq x \leq x_2(j,k) \\ \text{AND} \quad & y_1(k,i) \leq y \leq y_2(k,i) \\ \text{AND} \quad & z_1(j,i) \leq z \leq z_2(j,i) \end{aligned} \quad (2)$$

2.3. Domain of Applicability

The algorithm presented above is constrained by certain limitations imposed by the buffers themselves. Since the method uses data from the z-buffer, the information available about the object is limited to parts of the object visible from the three principal viewing axes. As a result the method is not able to correctly reconstruct objects with internal cavities or other parts, which can

not be seen from the chosen viewing angles. This does not constrain the method to convex objects; non-convex objects can be voxelized, as long as there are no “hidden” cavities (Figs. 5, 6, 7, 8).

The algorithm is obviously not optimized for hollow objects (as additional computation is needed to remove the inner voxels), while it is ideal for applications where the interior of the object should be occupied with voxels. In such cases, many existing algorithms first voxelize the outer surface and subsequently perform a volume-filling operation [Wang, Kaufman 94]. A variation of the method which voxelizes only the outer surface of an object will be described in Section 4.

When dealing with convex objects, the algorithm may be further simplified by using only one pair of buffers, decreasing floating point operations and comparisons by $2/3$, as all required information can be retrieved from a single view. Even certain non-convex objects, on which the location of all non-convex characteristics is restricted to certain views, can be modeled with less than three pairs of buffers, provided that the buffers are placed at the viewing angles of interest (e.g. a torus can be voxelized using the pair of buffers that is parallel to the torus base plane).

3. Advantages

The proposed algorithm has two major advantages: it is very simple and easy to implement and it is extremely fast. Voxelization of an object to a reasonable spatial resolution is accomplished in real time in an average machine, as will be demonstrated in the Test Results section, even without specialized hardware. Since graphics cards supporting hardwired z-buffers are very popular even in most home computer systems, the method can take advantage of hardware acceleration for free.

Another important feature of our algorithm is that it can be used for all types of 3D objects (polygonal, analytical or even volume data) without any modification or increase in computational cost, unlike many existing voxelization techniques which are restricted to specific types of objects [Kaufman, Shimony 86], [Jones 96]. The method can even be used for simultaneous voxelization of multiple objects, so long as there is no overlap resulting in hidden areas among them for all views.

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.

An additional advantage is that the algorithm can be used to easily produce multiresolutional models, simply by repeated execution for different buffer resolutions.

Finally, the method offers the possibility to develop volume

sculpting techniques based on the manipulation of the two-dimensional buffers, instead of voxel space operations [Galyean, Hughes 91], [Wang, Kaufman 95]. We are currently working in this direction.

4. Variations of the Basic Algorithm

We next describe two variations of the basic algorithm: one for voxelizing only the contour of an object and one for the generation of volume data of varying density. The second technique is suitable for convex objects; however it can be used to voxelize certain non-convex objects, with reduced accuracy.

4.1. Surface Voxelization

A voxel $v(i,j,k)$ is part of the object surface, if at least one of its coordinates, calculated by (Eq. 1), lies close to one of the corresponding buffer values, while the remaining coordinates satisfy the conditions in (Eq. 2). Therefore $v(i,j,k)$ lies on the surface if the following condition (or any of the two analogous conditions for Y and Z axes) is true :

$$\begin{aligned}
 & (\quad x_1(j,k) \leq x \leq x_1(j,k) + d_{x1}(j,k) \\
 & \quad \text{OR} \quad x_2(j,k) - d_{x2}(j,k) \leq x \leq x_2(j,k) \\
 &) \\
 & \text{AND} \quad y_1(k,i) \leq y \leq y_2(k,i) \\
 & \text{AND} \quad z_1(j,i) \leq z \leq z_2(j,i)
 \end{aligned} \tag{3}$$

where x, y, z are calculated from (Eq. 1) and d is a deviation threshold defining how close to the exact contour of the object a voxel should be. In other words, d is a measure of the surface thickness at each point with regard to the viewing axis. Surface thickness is measured from the object contour towards the object interior; that is why the above inequalities for x are not symmetrical.

The value of d must be chosen in a way that combines two objectives [Kaufman, Cohen, Yagel 93]:

- 1) The derived voxel set must cover the entire object surface, without leaving holes.
- 2) The number of voxels which are included in the final set should be minimised.

These conditions may not be served simultaneously with a single deviation threshold for the entire voxel space. Instead, it will be shown that optimal results can be obtained by defining a different value of d for each voxel with regard to each buffer (e.g. d_{x1}, d_{x2} etc).

As d denotes the surface thickness in a given direction (thickness depends on the surface orientation with regard to the viewing angle), its value must equal the maximum depth difference between subsequent voxels visible to a certain buffer.

Consider the example in Fig. 3, which is reduced to two dimensions for simplicity. $v(i,j)$ and $v(i+1,j+3)$ are successive visible voxels from the z_1 buffer (voxels marked with an “X” symbol) and correctly identified as surface voxels. However, voxels $v(i,j+e)$, $e=1..3$, which are also surface voxels (voxels marked with an “O” symbol), are hidden from the z_1 buffer. If these voxels are visible to another buffer (z_2 in this example) they will be correctly identified as belonging to the surface. This is not always the case, especially in non-convex objects (e.g. $v(i,j+2)$ is hidden to all buffers). To avoid this problem we calculate the surface thickness $d_{z_1}(i,j)$ for $v(i,j)$ as viewed from the z_1 buffer and mark all voxels $v(i,j+e)$, $e=1..d_{z_1}(i,j)$. These voxels belong to the object surface, provided the remaining criteria of (Eq. 3), concerning the remaining axes, are matched.

$d_{z_1}(i,j)$ can be directly derived from the z_1 buffer, by calculating the intensity differences between $z_1(i)$ and adjacent buffer points, which correspond to depth differences in object space :

$$d_{z_1}(i,j) = \begin{cases} z_{1\max}(i) - z_1(i), & \text{if } z_{1\max}(i) \geq z_1(i) \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where $z_{1\max}(i) = \max\{z_1(i-1), z_1(i+1)\}$.

If a smaller value of $d_{z_1}(i,j)$ is chosen, the contour will have holes, while a greater value will produce redundant voxels in the final model.

If processing the opposite buffer, z_2 , the comparison criterion will be reversed and $z_{2\min}$ will be calculated instead.

The same procedure is repeated for the two x buffers.

Extending the procedure to three dimensions is simple. The only difference is that to define the surface thickness, we need to examine 4 or 8 neighbors of a buffer point, depending on the desired connectivity.

Surface thickness calculations are performed as a preprocessing stage to the surface voxelization procedure. Computational complexity is analogous to the buffer dimensions ($O(N^2)$), which is not a significant overhead compared to the $O(N^3)$ complexity of the voxelization. The surface voxelization algorithm is given below:

Step1. Read the depth buffers ($x_1, x_2, y_1, y_2, z_1, z_2$).

Step2. For each buffer:

For each buffer cell:

Calculate the maximum depth difference of the cell from its neighbours. Store the result in a new “thickness” buffer $(dx_1, dx_2, dy_1, dy_2, dz_1, dz_2)$.

Step3. For every voxel $v(i,j,k)$ check if it is a surface voxel by using relation (3).

4.2. Varying Density Voxelization

Some volume visualization applications require images from volume data with more than two values per voxel, in order to reduce the “blockiness” effect present in binary decision methods (1 if the voxel is inside the object – 0 otherwise). In this case voxels are assigned a value which is a function of their distance from the object surface, or the smoothing effect is achieved via filtering.

Smoothing is in general useful on the external surface of the object, which is responsible for the jagged effect. Our method can easily be modified, for convex objects, to produce voxels of a varying density. A voxel’s density is estimated according to the percentage of the voxel that is actually inside the object. Exterior voxels have a density of 0, while voxels completely inside the object are assigned a density of d_{max} . Voxels on the contour have a density value proportional to the volume covered by the object. This percentage is calculated as follows:

Consider first the X axis. Buffer values $x_1(j,k)$ and $x_2(j,k)$ for a voxel $v(i,j,k)$ are retrieved and the result is converted to the corresponding values i_1 and i_2 , in the range $[0, N]$, in a way reverse to (Eq. 1):

$$i_{1,2} = x_{1,2} \frac{N}{depth_{max} - depth_{min}} \quad (5)$$

Note that buffer values are defined in the range $[depth_{min}, depth_{max}]$ and are therefore converted in the range $[0, N]$, instead of $[0, N]$, which justifies the presence of N on the numerator instead of $N-1$ which one would expect from (Eq. 1).

i (which represents the voxel coordinate on the X axis in voxel space) has an integer value in $[0, N]$, while i_1 and i_2 (which are the object surface depths on the X axis) are floating point numbers in $[0, N]$. Thus $v(i,j,k)$ is a surface voxel with regard to the X axis, if and only if $i = \lfloor i_1 \rfloor$ or $i = \lfloor i_2 \rfloor$.

- If $i = \lfloor i_1 \rfloor$, the X axis voxel coverage is set to $c_x = \lfloor i \rfloor + 1 - i_1$, while if $i = \lfloor i_2 \rfloor$ the corresponding coverage is $c_x = i_2 - \lfloor i \rfloor$ (Fig. 4).
- If $i < \lfloor i_1 \rfloor$ or $i \geq \lfloor i_2 \rfloor$ then $v(i,j,k)$ is an external voxel and we set $c_x = 0$.

- In all other cases, i lies between $\lfloor i_1 \rfloor$ and $\lfloor i_2 \rfloor$ and the coverage in the direction of the X axis is 1.

In the same way we define the coverage in the remaining two directions Y and Z , as c_y and c_z respectively. Total voxel density d_v is a combination of coverage percentages in all three directions :

$$d_v = c_x \cdot c_y \cdot c_z \cdot d_{\max} \quad (6)$$

5. Results

All algorithms presented in this paper were implemented in C++, using OpenGL under Windows98 on a 400MHz Intel Pentium II system, with a nVIDIA Riva TNT graphics board.

Table 1 shows the computational time for the voxelization of several polygonal objects, with respect to the number of faces of the original model, the buffer resolution and the number of voxels created. The first column displays the number of faces in the original triangular mesh, while the second gives the buffer resolution. The remaining columns show the number of voxels actually occupied and the time taken for simple voxelization, surface voxelization and density voxelization.

The table verifies that the computational time is extremely small even on an average machine. It depends only on the buffer resolution and it is independent of the number of faces in the original mesh.

In the case of density voxelization, there is an additional dependency on the percentage of voxels that are occupied (e.g. the time taken to voxelize the cross is greater than the time for the knot, since the percentage of voxels occupied for the former is significantly greater than the corresponding percentage for the later). This results from the fact that many calculations required for the density value, are omitted if a voxel is found early in the process to be outside the object.

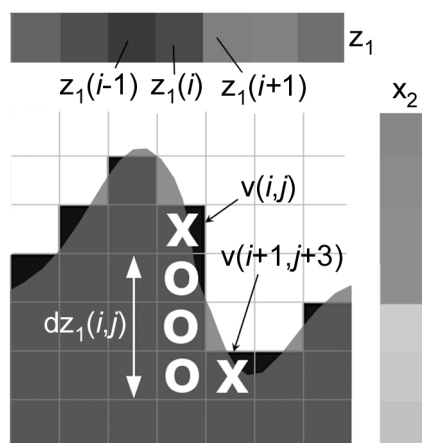
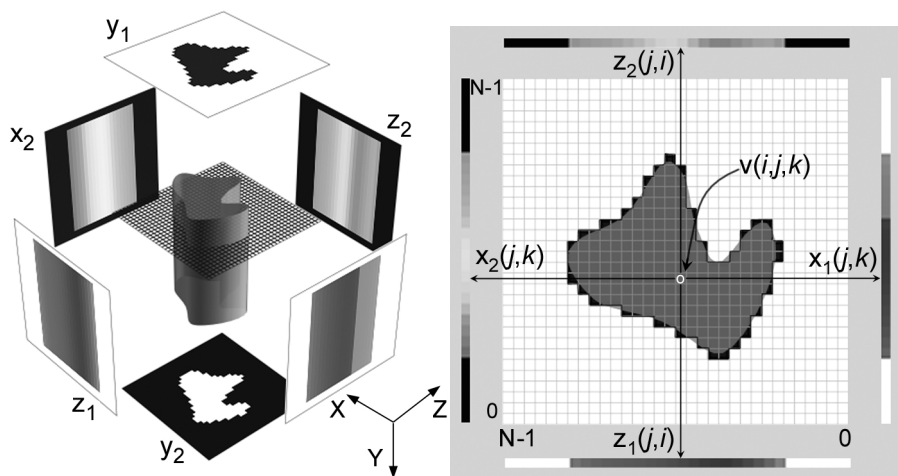
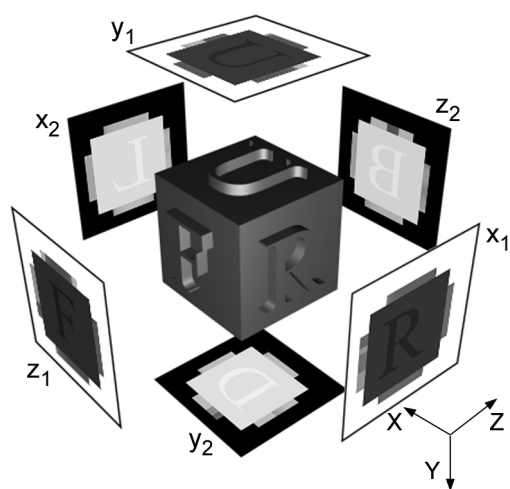
We should underline however, that the complexity depends only on the number of voxels calculated and retains the independency from the number of faces. This is an important advantage over other methods whose complexity depends on the number of faces in the original mesh and therefore become very time-consuming when used for complex meshes [Jones 96].

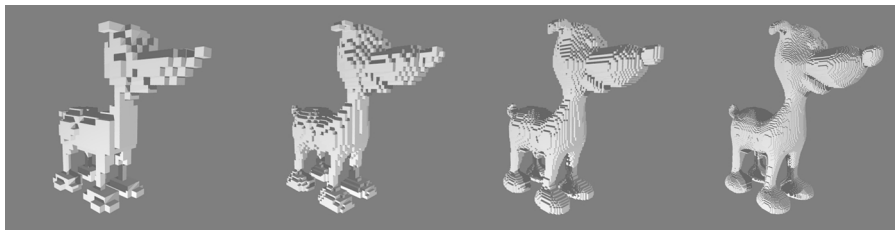
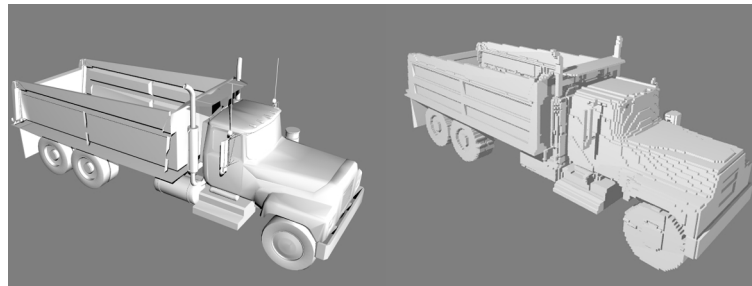
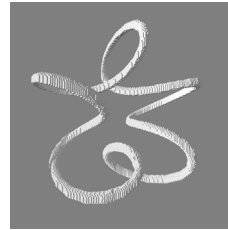
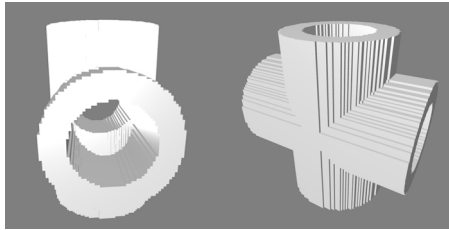
Figs. 5 a voxelized hollow cross, while Fig. 6 shows a knot created using surface voxelization. Fig. 7 displays a cartoon dog, voxelized using buffers of decreasing dimensions to produce the model at multiple resolutions. Fig.8 displays a voxelized truck model.

References

- [Cohen, Kaufman, Wang 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).
- [Cohen-Or 97] D.Cohen-Or. "Exact antialiasing of textured terrain models." *The Visual Computer*, 13:184-199 (1997).
- [Galyean, Hughes 91] T.Galyean and J.Hughes. "Sculpting: an Interactive Volumetric Modeling Technique." *Computer Graphics (Proc. SIGGRAPH '91)*, 25: 267-274 (1991).
- [Jones 96] M.Jones. "The Production of Volume Data from Triangular Meshes Using Voxelization." *Computer Graphics Forum*, 15:311-318 (1996).
- [Kajiya, Kay 89] J.T.Kajiya and T.L.Kay. "Rendering Fur with Three Dimensional Textures." *Computer Graphics (Proc. SIGGRAPH '89)*, 23:271-280 (1989).
- [Kaufman, Cohen, Yagel 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.
- [Neyret 98] F.Neyret. "Modeling, Animating and Rendering Complex Scenes Using Volumetric Textures." *IEEE Transactions on Visualization and Computer Graphics*, 4:55-70 (1998).
- [Pommert et al. 92] A.Pommert, M.Bomans, M.Riemer et al. "Volume Visualization in Medicine: Techniques and Applications." In *Focus on Scientific Visualization*, edited by H.Hagen, pp 41-71. Berlin: Springer, 1992.
- [Prakash, Manohar 95] C.E. Prakash and S.Manohar. "Volume Rendering of Unstructured Grids – A Voxelization Approach." *Computer Graphics*, 19(5):711-726 (1995).
- [Stytz, Frieder, Frieder 91] M.R.Stytz, G.Frieder and O.Frieder. "Three-dimensional Medical Imaging: Algorithms and Computer Systems." *ACM Computing Surveys*, 23:421-499 (1991).
- [Wang, Kaufman 93] S.Wang and A.Kaufman. "Volume Sampled Voxelization of Geometric Primitives." In *Proceedings of the Visualization '93 Conference*, pp 78-85. IEEE Computer Society Press, 1993.
- [Wang, Kaufman 94] S.Wang and A.Kaufman. "Volume-Sampled 3D Modeling." *IEEE Computer Graphics and Applications*, 14:26-32 (1994).

[Wang, Kaufman 95] S.Wang and A.Kaufman. “Volume Sculpting.” In *Symposium on Interactive 3D Graphics*, pp151-156. ACM Press, 1995.





	# of triangles	Buffer Resolution	Voxelization		Surface Vox/tion		Density Vox/tion	
			# of Voxels	Time (sec)	# of Voxels	Time (sec)	# of Voxels	Time (sec)
24-hedron	48	64 X 64	75156	0.06	26079	0.29	80740	0.63
		128 X 128	622789	0.50	114895	1.90	576687	4.80
		256 X 256	5026771	6.00	484901	15.00	4995637	41.00
Torus	576	64 X 64	13573	0.05	9409	0.29	14985	0.27
		128 X 128	112076	0.45	42353	1.90	117718	2.20
		256 X 256	910021	6.00	171203	15.00	903101	19.00
Knot	1520	64 X 64	3169	0.06	3147	0.34	3588	0.20
		128 X 128	22624	0.40	22327	2.00	25198	1.70
		256 X 256	180859	5.00	116694	15.00	195519	15.00
Cross	1530	64 X 64	44428	0.05	20536	0.28	45330	0.45
		128 X 128	361961	0.50	90574	1.90	369651	3.60
		256 X 256	2930701	6.00	367404	15.00	2962560	31.00
Letter cube	3472	64 X 64	117917	0.06	25457	0.28	125255	0.87
		128 X 128	984392	0.50	94387	1.90	998349	7.00
		256 X 256	7924242	6.00	403722	14.00	7947191	57.00
Truck	23357	64 X 64	4498	0.05	4331	0.29	4424	0.21
		128 X 128	32093	0.40	28084	1.90	35205	1.70
		256 X 256	267794	5.00	143424	14.00	282432	16.00