3D Line Voxelization and Connectivity Control

Daniel Cohen-Or
Tel-Aviv University

Arie Kaufman
State University of New York at Stony Brook

The connectivity of consecutive voxels along a 3D discrete voxelized line determines its final shape, penetration, accuracy, and speed. New algorithms generate exact, discrete lines and adaptively control their connectivity.

Voxelization algorithms that convert a 3D continuous line representation into a discrete line representation have a dual role in graphics. First, these algorithms synthesize voxel-based objects[1] in volume graphics.[2] The 3D line itself is a fundamental primitive, also used as a building block for voxelizing more complex objects. For example, sweeping a 3D voxelized line along a 3D voxelized circle generates a voxelized cylinder.[3]

The second application of 3D line voxelization algorithms is for ray traversal in voxel space. Rendering techniques that cast rays through a volume of voxels are based on algorithms that generate the set of voxels visited (or pierced) by the continuous ray. Discrete ray algorithms have been developed for traversing a 3D space partition[4-6] or a 3D array of sampled or computed data.[7] These algorithms produce one discrete point per step, in contrast to ray casting algorithms for volume rendering, which track a continuous ray at constant intervals, and to voxelization algorithms that generate nonbinary voxel values (for example, partial occupancies).[8]

Before considering algorithms for generating discrete lines, we introduce the topology and geometry of discrete lines.

Properties of discrete lines

Let Z^3 be the subset of the 3D Euclidean space R^3 that consists of all R^3 points whose coordinates are integers. This subset is called the *grid*. A *voxel* is a closed unit cube whose center is a grid point. Each grid point associates with a voxel. A surjective function maps Z^3, and hence the voxels, to {0, 1}. We assign the value 1 to nonempty voxels and value 0 to the others. This is therefore a point sampling process, which generates a binary voxelization. Nonbinary voxelization[8] generates superior volume-sampled smooth objects. However, binary voxelization, as discussed in this article, is still the first step in nonbinary voxelization and the only step in ray traversal.

Every voxel has 26 adjacent voxels: 8 share a corner (vertex) with the center voxel, 12 share an edge, and 6 share a face. Accordingly, we define face-sharing voxels as 6-adjacent, and edge-sharing or face-sharing voxels as 18-adjacent; two voxels are considered 26-adjacent if they share a vertex, edge, or face. The prefix N defines the adjacency relation, where $N \in \{6, 18, 26\}$ (in the 2D case, $N \in \{4, 8\}$). An *N-path* in $W \subseteq Z^3$ is a sequence of voxels all in W, such that consecutive pairs are N-adjacent. A set of voxels A is N-connected if an N-path exists between every pair of points in A.

An N-line is an N-path that represents in Z^3 a continuous line in R^3. The discrete representation of an N-line, like the representation of any other voxelized object, is an approximation of a continuous entity by a discrete set.[9] We refer to a discrete line as an N-path where the continuous line pierces all its voxels. This definition guarantees that the discrete line's voxels remain connected and close to the continuous line. However, the discrete line is not a unique representation of the continuous line. We have some flexibility in selecting the discrete line's voxels. We refer to a discrete line as the "best approximation" of a continuous line if the voxels of the discrete line are generated by an algorithm that incrementally selects one of its N-neighbors by minimizing the Euclidean distance from the continuous line.

This discrete representation, however, has several problems. Figure 1a shows a 2D 8-connected discrete line with an 8-tunnel (8-connected white path) passing from one side of the line to the other. To avoid such a scenario, convention defines opposite types of connectivities for the "1" and the "0" sets. In 3D, however, the situation is much more complex because the connectivity of a surface does not fully characterize its topology.[10,11] For example, an 18-connected surface might have holes or

0272-1716/97/$10.00 © 1997 IEEE

tunnels (see Figure 1b). As a consequence, a voxelized surface is characterized by the absence of tunnels. Loosely speaking, a tunnel is a discrete passage through a voxelized surface where no analogous passage exists in the continuous form.[11]

One practical meaning of a tunnel becomes apparent when a voxelized scene is rendered by casting discrete rays from the image to the scene. Penetration of a viewing ray through a tunnel in the voxelized object causes the appearance of a false hole in the object. Apparently, the rays can be implemented by a discrete line with any connectivity type. Specifically, a 26-ray might penetrate through 26 tunnels in a voxelized object that has only 6 tunnels free. However, a 26-ray does not traverse all voxels pierced by the continuous ray. Thus, it may skip a voxel in which the continuous ray meets the continuous object.[7] To avoid such skips, all the voxels pierced by the continuous ray have to be traversed by the discrete line. When a discrete line contains the entire continuous line it represents, then we say that the line has the *containment property*. In very rare cases, when a ray makes a 45-degree angle with all or some of the main axes, a 26- or an 18-connected line may have the containment property. However, for other cases, a discrete line with the containment property is 6-connected. Ray tracing uses discrete rays with the containment property,[4,5] but they're never classified as 6-connected rays.

An 18-ray resembles the 26-ray—it does not have the containment property and can penetrate opaque discrete objects. You might wonder about the usefulness of the 26-rays and the 18-rays. They're usually shorter and have fewer voxels than 6-rays and prove more attractive for ray traversal.

To understand the notion of the length of discrete lines, we now discuss the discrete metrics for the various connectivities. Here we define three metrics in Z^3, which correspond to the three connectivity types. The 6-distance d_6 between two voxels (x_1, y_1, z_1) and (x_2, y_2, z_2) is

$$d_6 = |\Delta x| + |\Delta y| + |\Delta z| \quad \textbf{(1)}$$

which is the length of the shortest 6-path between the two voxels, where $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$, and $\Delta z = z_2 - z_1$. The 26-distance d_{26} between two voxels is

$$d_{26} = \max\left(|\Delta x|, |\Delta y|, |\Delta z|\right) \quad \textbf{(2)}$$

which is the length of the shortest 26-path between them. The 18-distance d_{18} between two voxels is

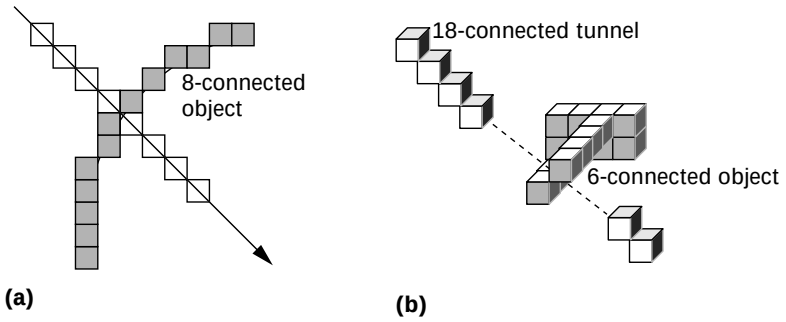
$$d_{18} = \max\left(d_{26}, \lceil d_6/2 \rceil\right) \quad \textbf{(3)}$$

which is the length of the shortest 18-path between the two voxels. It is widely known and quite easy to show that d_6 and d_{26} are metrics in Z^3. For more information, see the sidebar "Proof that d_{18} Is a Metric."

Equations 1 and 2 show that a 26-ray is shorter than a 6-ray by a factor of up to three:

$$1 \le \frac{d_6}{d_{26}} \le 3 \quad \textbf{(4)}$$

When the line is parallel to one of the primary axes, $d_6 = d_{26}$, and when the line approaches a major diagonal, the ratio d_6/d_{26} increases up to three.



1 (a) An 8-connected tunnel passes through an 8-connected object. (b) An 18-connected tunnel passes through a 6-connected object.

Proof that d_{18} Is a Metric

We have to show that d_{18} is also a metric in Z^3, namely, that it satisfies the axioms

1. $d_{18}(p_1, p_2) = d_{18}(p_2, p_1)$
2. $d_{18}(p_1, p_2) = 0$ iff $p_1 = p_2$
3. $d_{18}(p_1, p_3) \le d_{18}(p_1, p_2) + d_{18}(p_2, p_3)$

Axioms 1 and 2 are trivially established from the definition of d_{18} (Equation 3) written as a distance function between p_1 and p_2:

$$d_{18}(p_1, p_2) = \max(d_{26}(p_1, p_2), \lceil d_6(p_1, p_2)/2 \rceil)$$

Axiom 3, the triangle inequality, is based on the following inequalities:

$$\max(a, \lceil b \rceil) + \max(c, \lceil d \rceil)$$

$$= \max(a + c, a + \lceil c \rceil, \lceil b \rceil + c, \lceil b \rceil + \lceil d \rceil)$$
$$\ge \max(a + c, \lceil b \rceil + \lceil d \rceil)$$
$$\ge \max(a + c, \lceil b + d \rceil)$$

assuming $a, b, c, d \ge 0$.

Thus,

$$d_{18}(p_1, p_2) + d_{18}(p_2, p_3)$$
$$= \max(d_{26}(p_1, p_2), \lceil d_6(p_1, p_2)/2 \rceil)$$
$$+ \max(d_{26}(p_2, p_3), \lceil d_6(p_2, p_3)/2 \rceil)$$
$$\ge \max(d_{26}(p_1, p_2) + d_{26}(p_2, p_3),$$
$$\lceil d_6(p_1, p_2)/2 + d_6(p_2, p_3)/2 \rceil)$$

From the triangle inequality of d_{26} and d_6 we get

$$\ge \max(d_{26}(p_1, p_3), \lceil d_6(p_1, p_3)/2 \rceil)$$
$$= d_{18}(p_1, p_3)$$

Table 1. Experimental versus analytical average length ratios between discrete distances.

		d_6	d_{18}	d_{26}
d_6	Analytical	1	1.76	1.866
	Experimental	1	1.76	1.84
d_{18}	Analytical	0.58	1	1.053
	Experimental	0.57	1	1.05
d_{26}	Analytical	0.526	0.956	1
	Experimental	0.54	0.96	1

Table 2. Experimental average length of a discrete line in a 100^3 space.

Metric	Length
d_6	100.07
d_{18}	56.87
d_{26}	54.34

The analytical average length ratios between d_6, d_{18}, and d_{26} as shown in Table 1 have been computed by analytically determining their analytical limits.[12] The analytical results were compared with ratios obtained experimentally by randomly placing two voxels (the line endpoints) in a 100^3 voxel space. Table 2 shows the observed average distances. For more information, see the sidebar "Proofs for the Average Discrete Length." Note that the ratio of d_{18} and d_{26} is very close to one. In fact, 18-lines and 26-lines prove similar because they differ only at major diagonal moves, which are quite rare (occurring only for lines with both slopes close to 45 degrees).

The shortest path between two voxels is not necessarily the best approximation of the continuous line. A short 6-path between two voxels can step first only along x, then along y, and finally along z. However, a 6-path (26-path) that best approximates the continuous line is also a short path. In other words, it generates a 6-path (26-path) whose discrete length agrees with $d_6(d_{26})$. However, this is not true for 18-lines, as you can see in Figure 2. The shortest path of a continuous line from (0, 0, 0) to (2, 2, 2) takes three greedy steps (see Figure 2b). That is, it changes two directions at a time: (0, 0, 0), (1, 1, 0), (1, 2, 1), and (2, 2, 2). However, the 18-path (0, 0, 0), (1, 1, 0), (1, 1, 1), (1, 2, 1), and (2, 2, 2)—which is the "best approximation"—includes a local error-minimization step towards (1, 1, 1), which lies exactly on the continuous line (see Figure 2a).

Since the 6-distance and 26-distance agree with the best discrete approximation, they can be used to count down the discrete line length during its generation or traversal. This is a simple termination mechanism accomplished by a single machine instruction, an auto-decrement branch. This cannot, however, be used for 18-lines.

We have seen that 26- and 18-lines are more likely to penetrate through tunnels than 6-lines. They also might skip voxels pierced by the continuous line and miss an intersection that exists between the continuous ray and the object. On the other hand, 26- and 18-lines are shorter than 6-lines. In addition, when the former synthesize surfaces, they generate thinner voxelized surfaces. These surfaces contain fewer voxels relative to those generated with 6-lines, but they might have 26- and 18-tunnels. Plus the thinner surfaces do not have the necessary containment property for a proper simulation of the intersection of light rays with geometric objects.[6] Cast as discrete rays, 26- and 18-lines are generated faster than 6-rays. Even if the voxelized objects in the scene are 26-tunnel-free, 26- and 18-rays may occasionally miss the object surface near the silhouette because they do not contain the entire continuous line.

However, it is possible to alternate between connectivities, a technique that enjoys both the speed of 26-rays and the containment property of 6-rays with no penalty. We developed this technique (described in the section

Proofs for the Average Discrete Length

The following is a probability proof for the average discrete length of 6- and 26-lines (see Table 2). Let us consider the cubic continuous space $[0,1]^3$. A 3D line from (x_0, y_0, z_0) to (x_1, y_1, z_1) can result from a random and independent selection of six values $x_0, y_0, z_0, x_1, y_1, z_1$ in the interval [0, 1]. The size $|x_0 - x_1| = \Delta x$ is distributed with a density function $f(\Delta x = t) = 2 - 2t$. This is also the density function of Δy and Δz. Thus, the expected value of d_6 is

$$E(\Delta x + \Delta y + \Delta z) = 3E(\Delta x) = 3\int_0^1 (2 - 2t)t\,dt = 3\left[t^2\ 2\ /\ 3t^3\right]_0^1 = 1$$

The distribution function F of Δx is

$$\text{prob}\{\Delta x \le t\} = \int_0^t (2 - 2s)ds = 2t - t^2$$

and similarly for y and z. Then,

$$\text{prob}\{\max(\Delta x, \Delta y, \Delta z) \le t\} = \text{prob}\{\Delta x \le t\}\ \text{prob}\{\Delta y \le t\}\ \text{prob}\{\Delta z \le t\} = (2t - t^2)^3$$

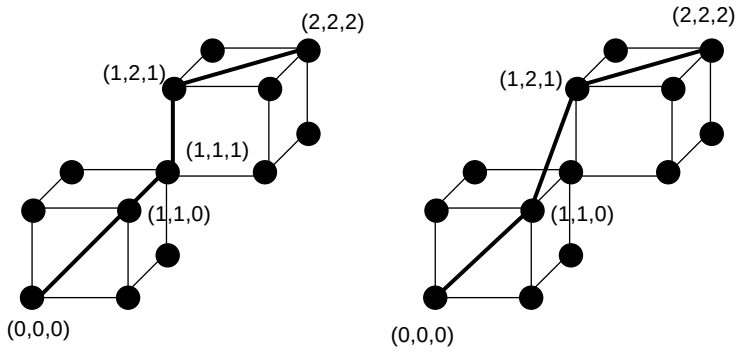
The density function of $\max(\Delta x, \Delta y, \Delta z)$ is the derivative

$$3(2t - t^2)(2 - 2t)$$

Thus, the expected value of d_{26} is

$$E(\max(\Delta x, \Delta y, \Delta z)) = \int_0^1 3(2t - t^2)(2 - 2t)dt = 19\ /\ 35 = .5428\ldots$$

See also our colleagues' work[12] for analytical computations of d_6, d_{18}, and d_{26}.



2 An 18-path representing a diagonal line from (0, 0, 0) to (2, 2, 2). (a) Best approximation by four steps. (b) The shortest path includes only three steps.

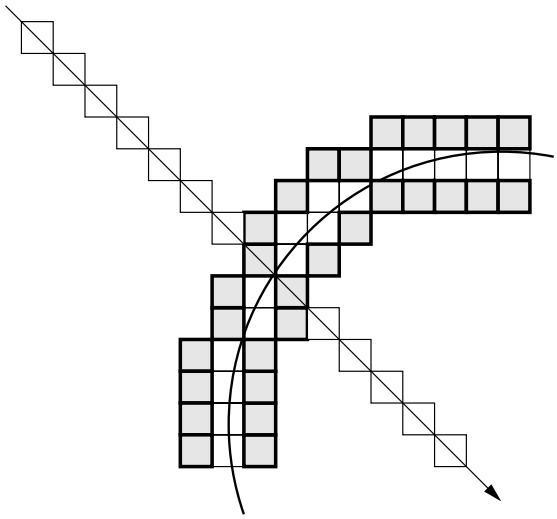
"Alternating between connectivities") for the discrete ray tracer,[6] where a proximity flag has been added to all the voxels around the object surface to indicate the vicinity of an object. In a nonbinary voxelized object,[8] the "fuzzy cloud" around an object can indicate proximity to the object. The cast ray starts as a 26-ray that skips rapidly over the empty regions. Whenever the ray encounters a proximity flag, the 26-line algorithm adaptively "slows down" and takes six connected steps to avoid misses at the object silhouettes or a leak through a tunnel. Figure 3 illustrates a 2D scenario where an 8-connected line hits one of the proximity flags surrounding the discrete object, which otherwise would have penetrated a tunnel.

The literature contains little on 18-connectivity. We saw above that the 18-line, like the 26-line, does not possess the containment property and is longer than the 26-line. We also showed that the best approximation would not necessarily provide the shortest path. In addition, while 6- and 26-connectivity are in many respects counterparts of 4-and 8-connectivity, 18-connectivity is a hybrid with no natural metric. However, 26-tunnel-free surfaces prove thicker than 18-tunnel-free surfaces. By employing 18-rays, we sacrifice very little speed (relative to 26-rays), yet reduce the time and space of the voxelization process. You'll find an 18-line algorithm in our earlier work.[3]

In the following section we present a 26-line algorithm that demonstrates the evolution of a 2D 8-connected-line algorithm to a 3D 26-line algorithm. The 26-line algorithm fits the structure of the 6-line algorithm developed in the section "The tripod 6-line algorithm" and thus enables the switching between the two connectivities, as described in the section "Alternating between connectivities."

A 26-line algorithm

Several well-known algorithms exist for the scan-conversion of 8-connected 2D lines (for example, the Bresenham line algorithm). They're based on an incremental algorithm that generates a sequence of discrete pixels, where at each step the next pixel is selected from two adjacent neighbors of the current pixel just drawn. The selected neighbor is the one that minimizes the error from the continuous line.[13] The midpoint algorithm[14] is an elegant way to choose the next pixel; for line drawings, it actually matches the Bresenham line



3 An 8-connected ray hits one of the proximity flags (in grey) surrounding the discrete surface, approximating the continuous surface (the black arc), which otherwise would have penetrated a tunnel.

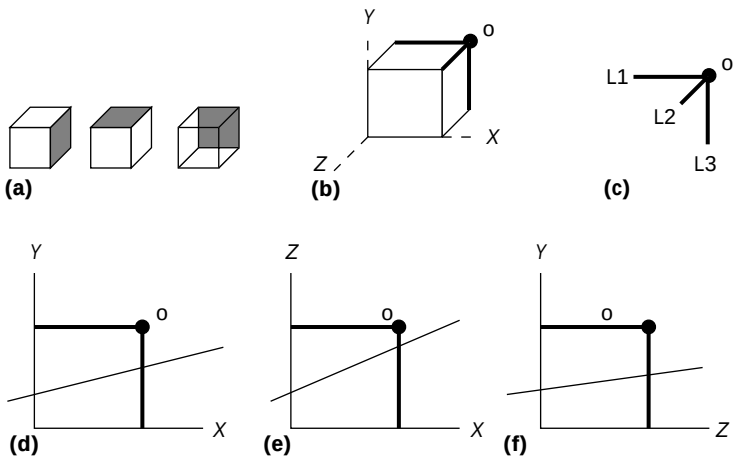
algorithm. The midpoint algorithm evaluates the residual error e at a midpoint between the two adjacent pixel candidates, where the sign of e indicates which of the two candidates lies closer to the continuous line.

Employing a first-order forward-differences technique, the value of e is updated incrementally by its first difference along the direction of the stepping. The 2D line is defined by its starting pixel at (x_0, y_0) and the last pixel at (x_0+A, y_0+B). Without degrading from the algorithm generality, we assume the line is a positive x-major (that is, $A \geq B \geq 0$). Then, x is incremented at every algorithm loop, which results in an asymmetric algorithm in x and y. To obtain a symmetric algorithm, you can use two decision variables, e_x and e_y, that determine whether or not to increment the x and y coordinates, respectively. The initial value of e_y is evaluated at the first midpoint (see Van Aken and Novak[14] for details).

Kaufman and Shimony[1] proposed a 3D line algorithm that is a 3D extension of the above 2D algorithm. It simultaneously generates two projections of the 3D line. Assume a 3D line from (x_0, y_0, z_0) to (x_0+A, y_0+B, z_0+C), where $A \geq B \geq C \geq 0$. The projection of the 3D line on the xy-plane is a 2D line from (x_0, y_0) to (x_0+A, y_0+B), and the projection on the xz-plane is a line from (x_0, z_0) to (x_0+A, z_0+C). Let dy_x, dy_y, and dy_{xy} be the forward differences along x, y, and the diagonal, respectively. Hereafter, we'll use similar notations for the forward

```
while (A—){
        SetVoxel (ptr);
        if (e_y < 0) {
              e_y += dy_x;
              if (e_z< 0){
                    e_z += dz_x;
                    ptr += offset_X;
                    }
              else {
                    e_z += dz_xz;
                    ptr += offset_XZ;
                    }
              }
        else {
              e_y += dy_xy;
              if (e_z < 0) {
                    e_z += dz_x;
                    ptr += offset_XY;
                    }
              else {
                    e_z += dz_xz;
                    ptr += offset_XYZ;
                    }
              }
        }
```

4 An efficient asymmetric 26-line algorithm.



5 The tripod concept: (a) the three candidate faces, (b) the tripod location, (c) the tripod apex and legs, (d)-(f) the tests of intersection with the tripod legs.

differences along other axes. The decision variable e_y on the xy-plane monitors the movement of the y coordinate, and similarly, e_z on the xz-plane monitors the movement of the z coordinate. The two independent decisions for the y and z axes are concatenated to yield an asymmetric 26-connected algorithm.[1]

We can significantly improve this algorithm by restructuring it to employ a tree of condition statements (Figure 4). Here, a pointer to the 3D array simplifies access to the voxels. The pointer is incremented by an offset according to the step direction. The algorithm presented in Figure 4 requires only three additions, two sign tests, and one auto-decrement-branch at each step, whereas Kaufman and Shimony's algorithm required up to five additions for a step with the same number of tests and auto-decrement-branches.

We presented symmetric 26-line algorithms elsewhere.[3] An algorithm is symmetric in the sense that it treats the three coordinates independently and uniformly, regardless of the major direction. Hence, it suits parallel computation and possibly parallel hardware implementation. The asymmetric algorithm presented here is more efficient, with fewer operations in the main loop and in the initialization step, but needs three copies, one for each major positive direction. (The negative directions are handled in a similar way.) The symmetric algorithm also forms the basis for the 6-line and 18-line algorithms we discussed previously.[3]

The tripod 6-line algorithm

Here we present a new, efficient 6-line algorithm—the tripod algorithm. This algorithm generates the 6-line by tracking the projections of the 3D line on the three main axes' planes. This algorithm guarantees the containment property and is more efficient than our 6-connected line algorithm.[3] Its efficiency is comparable to that of the parametric 6-connected line algorithms from Amanatides and Woo.[5]

Each voxel along the 6-line is face-adjacent to its predecessor. That is, the next voxel can be determined by the face the continuous line pierces when it leaves the voxels. Three faces then become candidates for the pierced face. Assuming a positive line (that is, $A \geq 0, B \geq 0, C \geq 0$), the continuous line departs from the current voxel (x, y, z) either through an $x + 0.5$ side face, a $y + 0.5$ upper face, or a $z + 0.5$ back face. The three faces, shown in Figure 5a, share a vertex marked by o in Figures 5b through 5f. The three edges emanating from o connect adjacent faces and form the shape of a tripod. The face from which the continuous line departs can be detected by testing the relation between the three tripod legs L1, L2, and L3 (Figure 5c) and the continuous line. In the following section we employ the midpoint technique to accomplish these tests to get the pierced face.

The projection of the 3D line on the xy plane introduces a line from (x_0, y_0) to $(x_0 + A, y_0 + B)$ with an implicit equation:

$$e_{xy} = Ay - Bx - D_1 = 0$$

Similarly, the projection of the 3D line on the xz plane is a line from (x_0, z_0) to $(x_0 + A, z_0 + C)$ with an implicit equation:

$$e_{xz} = Az - Cx - D_2 = 0$$

On the zy plane we get

$$e_{zy} = Bz - Cy - D_3 = 0$$

The above three 2D lines are defined on the three main

planes, which are perpendicular to the z-axis, y-axis, and x-axis, respectively. The code in Figure 6 is based on three such tests, shown in Figures 5d through 5f:

1. If ($e_{xy} < 0$), the next voxel is not y-adjacent.
2. If ($e_{xz} < 0$), the next voxel is not z-adjacent.
3. If ($e_{zy} < 0$), the next voxel is not y-adjacent.

Each of the above tests eliminates one of the tripod faces (Figure 5a). After the first test, the two remaining faces are tested by one of the other two tests. For example, a positive first test 1 indicates that the line does not leave the voxel from a y-adjacent voxel. Then, it must leave from either a z-adjacent voxel or an x-adjacent. A positive second test 2 indicates that the line does not depart from a z-adjacent voxel, implying that it must leave from an x-adjacent voxel. Other cases are similar.

Evaluating the decision variables is similar to evaluating the midpoint mechanism.[14] The midpoint location is equidistant from the two candidates at the corner of the pixel, such that its sign determines whether the line intersects a side edge or an upper edge.

Initializing the tripod algorithm in Figure 6 assumes that the line origin lies at the center of the voxel. Since the discrete form of the line is modulo its integer endpoint coordinates, its decision variables (e_{xy}, e_{xz}, and e_{zy}) can be initialized at (0, 0, 0), and the initial midpoint is located at (0.5, 0.5, 0.5). For example,

$$e_{xy}(0.5, 0.5) = 0.5A - 0.5B$$

To avoid fractions, we multiply the value of e_{xy} by two (see Figure 6). The algorithm can be extended to take subvoxel endpoint coordinates x, y by initializing the midpoint at (p_x, p_y), where $p_x = \lfloor x \rfloor + 1 - x$ for a positive x direction, and $p_x = x - \lfloor x \rfloor$ for a negative x direction (similarly for p_y). We also multiply the value of e_{xy} (p_x, p_y) by the subvoxel resolution to avoid fractions.

The 6-line tripod algorithm uses only two sign tests, three additions, and one auto-decrement-branch, plus a simplified access to the 3D array (exactly the same as in the efficient 26-line algorithm of Figure 4). Of course, it is more expensive to generate a 6-line than a 26-line, because about twice as many voxels have to be generated. However, the cost of generating one voxel remains the same for both lines. The 6-line algorithm we discussed previously[3] was not optimized and has as many as four additions more than the 6-line tripod algorithm. Plus, it does not guarantee to have the containment property.

The parametric 6-line algorithm

Amanatides and Woo[5] used a different approach to generate a 6-line for the voxel traversal algorithms. For the sake of completeness and for comparison with the tripod algorithm, we briefly describe it here. This approach uses a parametric line representation

$$\vec{X} + t\vec{V}$$

or

$$x_0 + tV_x \quad y_0 + tV_y \quad z_0 + tV_z$$

```
A2 = 2A; B2 = 2B; C2 = 2C;
exy = B − A;
exz = C − A;
ezy = B − C;
while (n−−) {
     SetVoxel (ptr);
     if (exy< 0) {
          if (exz < 0) {
               ptr += offsetx;
               exy += B2; exz += C2;
               }
          else {
               ptr += offsetz;
               exz −= A2; ezy += B2;
               }
          }
     else{
          if (ezy < 0) {
               ptr += offsetz;
               exz −= A2; ezy += B2;
               }
          else {
               ptr += offsetY;
               exy −= A2; ezy −= C2;
               }
          }
     }
```

6 The 6-line tripod algorithm.

Treating the parameter t as time, the portion of time that it takes to cross consecutive grid lines along the k direction is constant, denoted by ∂t_k. We denote the time when the line crosses the i^{th} k-grid line as $e_k(i)$. Conceptually, we get three sequences of crossing times:

$$e_x(0), e_x(1), \ldots e_x(i), \ldots$$
$$e_y(0), e_y(1), \ldots e_y(i), \ldots$$
$$e_z(0), e_z(1), \ldots e_z(i), \ldots$$

Since $e_k(i) - e_k(i+1) = \partial t_k$, the sequences can be calculated incrementally. Merge sorting the three sequences gives the order of crossing between adjacent voxels. Actually, the generation of the time sequences together with the merge sorting are performed incrementally on the fly.

Figure 7 (next page) shows the 6-line parametric algorithm. Assuming fixed-point arithmetic, it requires two comparisons, two additions, and one auto-decrement-branch. To compare its efficiency with the tripod algorithm, the comparison instructions decompose into a subtraction (addition) followed by a sign test. Thus, the 6-line parametric algorithm requires four additions, two sign tests, and one auto-decrement-branch—one addition more than the 6-line tripod algorithm described in the previous section. If the sign test is as fast as a general comparison test, the parametric algorithm requires fewer instructions. However, the initialization part of the parametric algorithm involves expensive computations (including square root and divisions) when generating the direction cosines (the normalization of the vector (A, B, C)) and the ∂t_k values. The contribution of the initialization part of the algorithm becomes significant when

```
while (n—) {
        SetVoxel (ptr);
        if (ex > ey) {
              if (ex > ez) {
                    ex += ∂tx;
                    ptr += offsetx;
                    }
              else {
                    ez += ∂tz;
                    ptr += offsetz;
                    }
              }
        else {
              if (ey > ez) {
                    ey += ∂ty
                    ptr += offsety;
                    }
              else {
                    ez += ∂tz
                    ptr += offsetz;
                    }
              }
        }
```

7 A 6-line parametric algorithm.

```
e8 = 2B – A;
dxy = 2B – 2A;
dx = 2B;
dy = –2A;
n = A;
while (n—) {
        if the pixel at (ptr) is
        nonempty goto alternate;
        if (e8 < 0) {
              e8 += dx;
              ptr += offsetx;
              }
        else {
              e8 += dxy;
              ptr += offsetxY;
              }
        }

alternate:

e4 = e8 – B;
while (x < A) {
        ReadPixel (ptr);
        if (e4 < 0) {
              ptr += offsetx;
              }
        else {
              e4 += dy;
              ptr += offsetY;
              }
        }
```

8 An 8-ray stepping algorithm with alternation to a 4-ray stepping.

many short lines (or rays) are generated.

Alternating between connectivities

When the discrete lines serve as viewing rays into a 3D discrete voxel space, as described above, it's possible to capitalize on both the speed of the 26-rays and the accuracy of the 6-rays. For example, this can be accomplished by marking the voxels in the vicinity of an object with a proximity flag, which indicates that the 26-ray should switch its connectivity to a 6-ray. Here, we discuss the mechanism to alternate between connectivities.

The 6-line algorithm uses decision variables, which are calculated at different locations than those in the 26-line algorithm. However, the locations of the two types of decision variables remain half a unit apart, and the value of one can easily be corrected to that of the other. Since our decision variables operate on the projection planes, we'll first demonstrate the idea for the 2D case.

Let $f(x,y) = Bx - Ay$ be the implicit equation of the line from (x_0, y_0) to $(x_0 + A, y_0 + B)$. A midpoint error variable e_8, calculated at $(x+1, y+0.5)$, guides the generation of an 8-connected line. The midpoint e_4 is calculated at $(x+0.5, y+0.5)$:

$$e_8 = f(x+1, y+0.5) \quad \textbf{(5)}$$

$$e_4 = f(x+0.5, y+0.5) \quad \textbf{(6)}$$

Since

$$\begin{aligned} &f(x+\Delta x, y) \\ &= B(x+\Delta x) - Ay \\ &= f(x,y) + \Delta x B \end{aligned} \quad \textbf{(7)}$$

and $\Delta x = 0.5$, we get

$$e_8 = e_4 + 0.5B \quad \textbf{(8)}$$

To avoid using fractions, we scale up the decision variables by a factor of two. The forward difference values are constants and easy to generate. The algorithm in Figure 8 traverses the pixel space, tracking a line from (0, 0) to (A, B) where $A > B$. The initial connectivity is 8. When the discrete line encounters a nonempty pixel (either an object pixel or a proximity flag), it adaptively "slows down" and steps as a 4-connected line. A similar algorithm can switch between 4-line and 8-line when the ray leaves the neighborhood of an object.

To show the extension of a 2D switching to a 3D switching between 26-connectivity and 6-connectivity, we use the notation of the 26-line algorithm of Figure 4 and the 6-line algorithm of Figure 6:

$$e_{xy} = e_y - 0.5B \quad \textbf{(9)}$$

$$e_{xz} = e_z - 0.5C \quad \textbf{(10)}$$

Since the 26-line algorithm does not use a midpoint on the zy-plane, the decision variable e_{zy} needs to be initialized by

$$e_{zy} = B(z + 0.5 - z_0) + C(y + 0.5 - y_0) \quad \textbf{(11)}$$

where y_0 and z_0 are the starting y and z coordinates of the line.

From Equations 9 and 10 we can derive the switching

from a 6-line algorithm to a 26-line algorithm:

$$e_y = e_{xy} + 0.5B \quad \textbf{(12)}$$

$$e_z = e_{xz} + 0.5C \quad \textbf{(13)}$$

Conclusion

We developed a C library of line voxelization functions. Discrete lines are classified by their connectivity: 6, 18, and 26, with accuracy-speed tradeoffs for these connectivity classes. As part of this library, we implemented our efficient algorithms for generating fast 26-lines and "accurate" 6-lines, including the Tripod algorithm and the adaptive technique that switches between connectivities, capitalizing on the advantages of both connectivities. This adaptive ray-traversal algorithm has operated successfully as the core of our discrete ray-tracer (RRT).[6] We are currently working on extending the tripod concept to the voxelization of other geometric primitives such as circles, ellipses, and parametric curves. ■

Acknowledgments

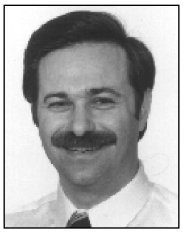
This project has been partially supported by the National Science Foundation under grants CCR-9205047 and MIP-9527694, and grants from the Office of Naval Research (N00014-97-10402, N00014-97-0362), Naval Research Lab (N00014-96-1-G015), and Hewlett-Packard.

We wish to acknowledge the careful review and helpful comments of Pat Hanrahan, Roger Hersch, Oscar Figueiredo, and Craig Gotsman. Many thanks to Sunmin Park for implementing the algorithms in this article.

References

1. A. Kaufman and E. Shimony, "3D Scan-Conversion Algorithms for Voxel-Based Graphics," *Proc. 1986 Workshop on Interactive 3D Graphics*, ACM Press, New York, 1986, pp. 45-75.
2. A. Kaufman, D. Cohen, and R. Yagel, "Volume Graphics," *Computer*, Vol. 26, No. 7, July 1993, pp. 51-64.
3. D. Cohen and A. Kaufman, "Scan-Conversion Algorithms for Linear and Quadratic Objects," A. Kaufman, ed., *Volume Visualization*, IEEE Computer Society Press, Los Alamitos, Calif., 1991, pp. 280-301.
4. A. Fujimoto, T. Takayu, and K. Iwata, "ARTS: Accelerated Ray-Tracing System," *IEEE Computer Graphics and Applications*, Vol. 6, No. 4, April 1986, pp. 16-26.
5. J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," G. Marechal, ed., *Proc. Eurographics 87*, Amsterdam, Elsevier Science B.V., 1987, pp. 3-9.
6. R. Yagel, D. Cohen, and A. Kaufman, "Discrete Ray Tracing," *IEEE Computer Graphics and Applications*, Vol. 12, No. 5, Sept. 1992, pp. 19-28.
7. J. R. Mitchell, "A Comparison of Line Integral Algorithm," *Medical Physics*, 1990, pp. 166-172.
8. S. Wang and A. Kaufman, "Volume-Sampled 3D Modeling," *IEEE Computer Graphics and Applications*, Vol. 14, No. 5, 1994, pp. 26-32.
9. C.L. Kim, "Three-Dimensional Digital Line Segments," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 5, No. 2, 1983, pp. 231-234.
10. D.G. Morgenthaler and A. Rosenfeld, "Surfaces in Three-Dimensional Digital Images," *Information and Control*, Vol. 51, 1981, pp. 227-247.
11. D. Cohen-Or and A. Kaufman, "Fundamentals of Surface Voxelization," *CVGIP: Graphics Models and Image Processing*, Vol. 56, No. 6, Nov. 1995, pp. 453-461.
12. S. Skiena, S. Balakrishnan, and C. Xu, *Length Ratios for Discrete Distance Metrics*, Tech. Report 92-08, Dept. of Computer Science, New York, State Univ. of New York (SUNY) at Stony Brook, Aug. 1992.
13. M.D. McIlroy, "Best Approximate Circles on Integer Grids," *ACM Trans. Graphics*, Vol. 2, No. 4, Oct. 1983, pp. 237-263.
14. J.R. Van Aken and M. Novak, "Curve-Drawing Algorithms for Raster Displays," *ACM Trans. Graphics*, Vol. 4, April 1985, pp. 147-169.

Daniel Cohen-Or *is a senior lecturer at the Department of Computer Science at Tel-Aviv University. His research interests include rendering techniques, volume visualization, architectures, and algorithms for voxel-based graphics. He received a BS cum laude in mathematics and computer science (1985) and an MS cum laude in computer science (1986) from Ben-Gurion University, Israel. In 1991 he received his PhD from the Department of Computer Science at State University of New York at Stony Brook.*



Arie E. Kaufman *is the director of the Center for Visual Computing and a professor of computer science and radiology at the State University of New York at Stony Brook. His research interests include volume visualization, graphics architectures, algorithms, languages, user interfaces, and multimedia. He received a BS in mathematics and physics from the Hebrew University of Jerusalem in 1969, an MS in computer science from the Weizmann Institute of Science, Rehovot, Israel in 1973, and a PhD in computer science from Ben-Gurion University, Israel, in 1977.*

Kaufman is currently editor-in-chief of IEEE Transactions on Visualization and Computer Graphics. *He has served as papers or program co-chair for the IEEE Visualization conference from 1990 to 1994 and was a co-founder and member of the steering committee of that conference series. He received the 1995 IEEE Outstanding Contribution Award and the 1996 IEEE Computer Society Golden Core Member Award.*

Contact Kaufman at the Center for Visual Computing, Dept. of Computer Science, State University of New York, Stony Brook, NY 11794-4400, e-mail ari@cs.sunysb.edu.

Contact Cohen-Or at the Dept. of Computer Science, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel, e-mail daniel@math.tau.ac.il.