Eurographics Symposium on Rendering 2013
Nicolas Holzschuch and Szymon Rusinkiewicz
(Guest Editors)

Volume 32 (2013), Number 4

A Topological Approach to Voxelization

Samuli Laine

NVIDIA

Abstract

We present a novel approach to voxelization, based on intersecting the input primitives against intersection targets in the voxel grid. Instead of relying on geometric proximity measures, our approach is topological in nature, i.e., it builds on the connectivity and separability properties of the input and the intersection targets. We discuss voxelization of curves and surfaces in both 2D and 3D, and derive intersection targets that produce voxelizations with various connectivity, separability and thinness properties. The simplicity of our method allows for easy proofs of these properties. Our approach is directly applicable to curved primitives, and it is independent of input tessellation.

1. Introduction

Voxelization is the process of producing a discrete 3D representation of an object, similar to rasterization that operates in 2D. The resulting data structure is useful across a range of applications, including global illumination, collision detection, visualization, and simulation. See, e.g., [DCB*04, ED06, Pan11] for pointers to numerous applications.

There are several possible definitions of a "proper" voxelization that depend on the dimensionality of the input geometry (points, lines, triangles, etc.), and the properties required by the subsequent application (connectivity, separability, thinness, etc.). However, no general framework exists for describing these different schemes in common terms.

In this paper, we introduce a general formulation of voxelization that allows both solid reasoning about the topological properties of the results and efficient implementation. Our key idea is that of a geometric *intersection target* associated with each voxel: a voxel is included in the output if the input primitives intersect its target. We show, with proofs, that choosing the intersection targets appropriately results in different voxelizations with desired connectivity and separability properties. Our formulation subsumes many previous voxelization algorithms as special cases.

Our method can be applied to objects of varying effective dimension (to be defined later) and works in both 3D and 2D. Notably, it is independent of input tessellation, and supports curved primitives with no extra considerations. Techniques for efficiently enumerating the voxels for which the intersection tests are to be performed can be derived based on previous literature, where considerable effort has been put to finding computationally efficient solutions on various kinds of hardware [FC00, ED06, ZCEP07, SS10, Pan11].

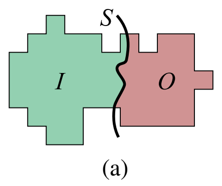
In addition to 3D voxelization, we shall examine 2D rasterization in this paper for two reasons. First, many key concepts are easiest to introduce in 2D and can then be extended to 3D. Second, our 2D analysis has direct applications in the rasterization of lines and curves with specific connectivity and separability requirements.

1.1. Basic definitions

Our notation mostly follows Cohen-Or and Kaufman [COK95]. Let $\mathbb{Z}^3$ be the set of 3D points with integer coordinates. We associate voxel $V = (x,y,z) \in \mathbb{Z}^3$ with the set of points $p = (x,y,z) \in \mathbb{R}^3$ where $V_x \leq p_x \leq V_x + 1$, and similarly for y and z. Let us consider a continuous, simple, and closed 2D-manifold surface S embedded in $\mathbb{R}^3$ so that $\mathbb{R}^3 - S$ has exactly two connected components, I and O. Let I^d and O^d be the nonempty sets of voxels that are entirely contained in I and O, respectively.

A voxelization of surface S is a discrete set of voxels S^d that has no common elements with I^d or O^d. S^d is said to be k-separating if there is no k-connected path of voxels $\Pi_k = (V_0, \ldots, V_n)$ where $V_{i+1} \in N_k(V_i)$ so that $V_0 \in I^d$, $V_n \in O^d$, and $\Pi_k \cap S^d = \emptyset$. In other words, S^d is a k-separating voxelization of S if every Π_k between I^d and O^d contains a voxel in S^d. $N_k(V)$ is the set of k-neighbors of voxel V, and in $\mathbb{Z}^3$ the neighbor sets of interest to us are

$$N_6(x,y,z) = (x \pm 1, y, z) \cup (x, y \pm 1, z) \cup (x, y, z \pm 1)$$



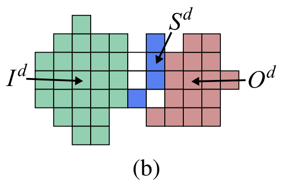


Figure 1: *Extending the notion of k-separability to non-closed surfaces. (a) Consider a non-closed surface S for which there exists a subset $\mathbf{z} \subset \mathbb{Z}^3$ such that S still separates the corresponding volume in $\mathbb{R}^3$ into two connected components I and O, and the corresponding I^d and O^d are nonempty. (b) S^d can be said to be a k-separating voxelization of S if every Π_k between I^d and O^d, restricted to voxels in $\mathbf{z}$, contains a voxel in S^d, and this holds for any valid choice of subset $\mathbf{z}$. In this 2D example, S^d is 4-separating but not 8-separating.*

$$
\begin{aligned}
N_{26}(x,y,z) &= \{x-1,x,x+1\} \times \{y-1,y,y+1\} \times \\
&\quad \{z-1,z,z+1\} - (x,y,z)
\end{aligned}
$$

where $\times$ denotes a cartesian product. As such, $N_6(V)$ has six elements and $N_{26}(V)$ has 26 elements for any V. A 6-connected path Π_6 is thus one where it is only allowed to walk into a neighboring voxel through voxel faces, and in a 26-connected path Π_{26} diagonal walks are allowed as well. Due to space constraints, we shall not consider 18-separating or 18-connected voxelizations or paths in this paper.

In 2D voxelization the situation is analogous, and in this case we use $S \subset \mathbb{R}^2$, $S^d \subset \mathbb{Z}^2$, V, I, O, and Π_k in their corresponding meanings. The dimension will be apparent from context. The neighborship relations in 2D are

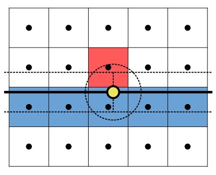
$$
\begin{aligned}
N_4(x,y) &= (x \pm 1, y) \cup (x, y \pm 1) \\
N_8(x,y) &= \{x-1,x,x+1\} \times \{y-1,y,y+1\} - (x,y)
\end{aligned}
$$

where N_4 is analogous to N_6 in 3D, and N_8 is analogous to N_{26}. Illustrations of the neighbor sets can be found in, e.g., [HYFK98].

We can extend the notion of separability to non-closed surfaces by considering a subset of voxel space $\mathbb{Z}^3$ such that the surface still separates the corresponding volume in $\mathbb{R}^3$ into two connected components, see Figure 1. Note that with both closed and non-closed surfaces, we restrict ourselves to cases where the corresponding I^d and O^d are nonempty. If this is not the case, it makes no sense to speak of separability because there is nothing to separate in $\mathbb{Z}^3$.

2. Previous work

Kaufman and Shimony [KS87] present algorithms for voxelizing various types of primitives, including line segments, polygons, and curved surfaces. They explicitly list a number of natural fidelity and connectivity requirements that we shall also follow in this paper. In particular, connectivity is



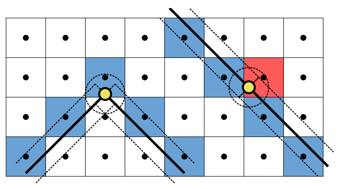


Figure 2: *Huang et al.'s method [HYFK98] for voxelizing triangle meshes (cf. polylines in 2D as illustrated here) is not minimal and it is sensitive to tessellation. (a) As noted in [SS10], in their 26-separating voxelization (cf. 8-separating in 2D), the use of edge cylinders and vertex spheres with $R_c = (\sqrt{3}/2)L$ may cause extraneous voxels to be tagged near the edges/vertices of a flat surface. (b) In addition, in 6-separating voxelization (cf. 4-separating in 2D), $R_c = L/2$ is the minimal radius that guarantees separation as shown on the left (adapted from Fig. 13 in [HYFK98]), but it may also produce extraneous voxels on flat surfaces, as shown on the right.*

the defining requirement for the fidelity of a voxelized 1D object, whereas 2D objects must meet a "lack of tunnels" requirement that is closely related to separability. Cohen-Or and Kaufman [COK97] further elaborate on fulfilling specific connectivity requirements for voxelizing 3D lines.

Cohen-Or and Kaufman [COK95] lay the ground for precise treatment of surface voxelization in 3D. They show that the so-called *supercover* of S, consisting of all voxels that meet S, is k-separating for any $k \in \{6, 18, 26\}$. The nature of the proof is similar to the ones used in this paper for voxelizations other than the supercover. They also discuss at length the proper handling of degenerate cases where S meets only the boundary face, edge, or vertex of a voxel. We present a simpler and more general solution to such degeneracies in this paper.

The above paper also introduces the concept of *tunnel-free* voxelization. This is a geometric property guaranteeing that certain connected paths in $\mathbb{R}^3$ outside S^d do not intersect the input surface S. Our 4-separating and 6-separating voxelizations in 2D and 3D are tunnel-free, and although we focus mainly on separability, we could easily obtain other tunnel-free voxelizations. This will be discussed in Section 5.2.2.

Wang and Kaufman [WK93] consider volume sampled voxelization, where a voxel's state is decided by intersecting a weighted spherical volume against input primitives, essentially prefiltering the input in 3D. This is done to combat aliasing artifacts that make, e.g., determining precise surface location in voxelized data difficult. The result of volume sampling can be interpreted as a distance field that allows a higher-fidelity reconstruction of the surface. Volume sampling has a connection to our intersection targets, although our goal is not to prefilter the input.

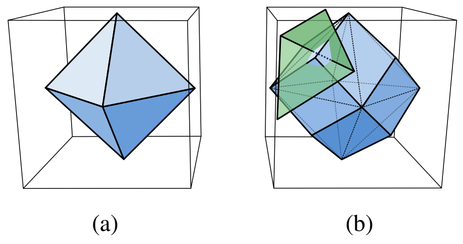


Figure 3: *The 6-separating surface voxelization method of Schwarz and Seidel [SS10] is equivalent to a rather peculiar series of geometric tests in 3D. (a) They first perform a plane test, which corresponds to intersecting the triangle's plane against an octahedron contained in the voxel. If a voxel passes this test, each axis-aligned 2D projection of the triangle is tested against a 2D diamond spanned between midpoints of pixel edges. (b) The intersection of the extruded 2D diamonds in 3D is a rhombic dodecahedron that completely encloses the octahedron used in the plane test. Pictured is a case where the mesh passes between these polyhedra so that the triangle planes intersect the octahedron but the triangles only intersect the rhombic dodecahedron. The final result can therefore be seen as an approximate intersection against the octahedron.*

Huang et al. [HYFK98] discuss the accuracy of voxelizing polygon meshes with the goal of producing minimal 6-separating and 26-separating voxelizations. They prove the minimality for 2D lines and 3D planes, and extend these results to 3D meshes composed of triangles. However, the edges and vertices of the triangles are handled in a conservative fashion, making the method sensitive to the tessellation of the input mesh and breaking the minimality. This was noted by Schwarz and Seidel [SS10] for the 26-separating case, and as illustrated in Figure 2, the special handling of edges and vertices also breaks minimality for the 6-separating case. Widjaya et al. [WME03] present an extension to general 2D and 3D lattices with separability and minimality proofs for infinite lines and planes.

Schwarz and Seidel [SS10] consider efficient GPU voxelization of triangle meshes based on axis-aligned 2D projections of the input triangles. They consider both 6-separating and 26-separating voxelizations, the latter being a *cover* of the input surface, i.e., a set of voxels so that every point in S is covered by a voxel in S^d. Their 6-separating voxelization is interesting because of its nontrivial geometric equivalent due to the 2D tests, and it can be seen as an conservative approximation of applying our voxelization scheme with an octahedral intersection target. See Figure 3 for further details. We show the separability and non-minimality of this method in Section 5.2.2. Pantaleoni [Pan11] presents a particularly efficient multi-stage GPU algorithm for both 6-separating and 26-separating voxelization of triangle meshes, also producing a cover as a solution to the latter case. Varadhan et al. [VKK*03] produce a cover by calculating the max-norm distance between voxel center and input geometry, and using this for determining whether an intersection occurs or not.

3. Voxelizing with intersection targets

In our method, we choose whether a voxel V is included in the voxelization S^d by testing if the input geometry intersects an *intersection target* contained within V. In all cases, it is enough to consider the voxels that belong to the supercover [COK95] of S, as the intersection targets in other voxels obviously cannot intersect the input geometry.

In the remainder of this section, we discuss how the effective dimension of the input should affect voxelization strategy, and how to handle degenerate intersections in a general way. In Section 4 we derive intersection targets for curves in 2D, and in Section 5 we discuss the voxelization of curves and surfaces in 3D. Finally, in Section 6 we examine variants where intersection targets may be different in each voxel.

3.1. Effective dimension of input

The intersection targets may be composed of primitives of various dimensions, see Figure 4 for examples. The appropriate choice of intersection target dimension depends on the *effective dimension* of the input, which may be different from the dimension of the input primitives. For example, consider a thin strand of hair that is geometrically represented as a solid tube. Even though the object is a volumetric solid, it is probably not a good idea to voxelize it using, e.g., the simple point-in-volume test, which in our terminology would correspond to the intersection target being a zero-dimensional point at the center of the voxel. Because the input is effectively one-dimensional, and the intersection target is zero-dimensional, an intersection happens only with arbitrarily small probability depending on the thickness of the hair. This results in voxelization that is not representative of the input, failing to preserve, e.g., its connectivity.

In this example situation, if we want to establish, e.g, 6- or 26-connectivity for S^d, we must use an intersection target that is at least two-dimensional, because then the intersection between the input and the intersection targets does not rely on chance. On the other hand, a three-dimensional intersection target (e.g., the entire voxel) would be unnecessarily

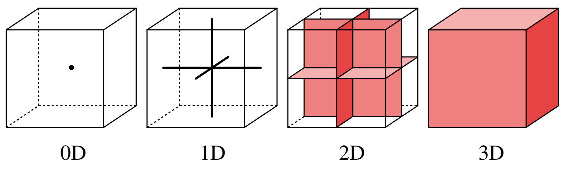


Figure 4: *Examples of intersection targets composed of primitives of various dimensions, suitable for different kinds of input.*

blunt and could result in redundant voxels being included in S^d. The same argument applies regardless of whether the hair is composed of a 3D solid, a 2D surface, or a 1D polyline or curve—the effective dimension is more relevant than the dimension of the input primitives.

In 2D voxelization, a similar situation occurs when a sequence of 2D primitives forms an object that is effectively one-dimensional. One such example would be a curve composed of segments with nonzero but small thickness, a situation common in path rendering. Rasterizing the path with the usual zero-dimensional intersection target (center of pixel) cannot guarantee connectivity of the result, but by placing a suitable 1D intersection target within each pixel, we can easily guarantee either 4- or 8-connectivity. Again, a full 2D intersection target would be excessive for these purposes.

The methods presented in previous literature are strictly specialized for the dimensionality of the input primitives, and as such cannot be applied to inputs where the primitive dimension happens to be (unnecessarily) higher than the effective dimension. For example, a line voxelization algorithm cannot be used for a thin hair composed of a 2D surface, even if we knew that the input were effectively one-dimensional. Our intersection target method does not suffer from this problem, and can handle inputs composed of primitives of any dimension within the same framework.

3.2. Degenerate intersections

The boundaries of input primitives and intersection targets must be handled correctly in order to avoid generating spurious holes in the voxelization, and to avoid unnecessarily "thick" results. Always including the boundaries (surfaces of volumes, edges of surface elements like triangles, endpoints of line or curve segments) in the intersection will prevent holes from forming, but when the input touches a boundary of multiple intersection targets, all of the corresponding voxels are included in S^d. Cohen-Or and Kaufman [COK95] discuss these problems at length, and propose using intersection targets where carefully selected parts of the boundaries of 2D pixels or 3D voxels are excluded.

Such a method is difficult to extend to more complex intersection targets, and we therefore suggest a more general and simpler solution to the problem of degenerate intersections. We first observe that any such degeneracy can be resolved by perturbing either the input geometry—or equivalently, the intersection target—slightly to an appropriate direction. This results in either no intersection, or a proper intersection of the input and the intersection target, where it does not matter whether boundaries are included or not.

It turns out that we can choose *a priori* an infinitesimal perturbation vector that removes any degeneracies between the input primitives and intersection targets. Applying this perturbation to the entire input—or equivalently, all intersection targets—produces a voxelization that is correct assuming that infinitesimal modifications to the input are allowed. An example of such a perturbation vector is $(\epsilon, \epsilon^2, \epsilon^3)$. In practice, a degenerate intersection is resolved by testing if it becomes non-degenerate by shifting the input (or the intersection target) infinitesimally, first in $+x$, then in $+y$, etc., until a non-degenerate result is obtained.

In voxel-vs-triangle and pixel-vs-edge tests, this approach is equivalent to the "reduced-voxels" of Cohen-Or and Kaufman [COK95], but does not require separately choosing which faces, edges, and vertices of the voxel boundary to include and which to exclude. In the point-vs-triangle test used in standard 2D rasterization, it is equivalent to the rasterization rules used in common graphics APIs, which guarantee that a fan of triangles with consistent facing covers each pixel center exactly once, even if it lies on an edge or a vertex. In practice, we expect that the infinitesimal perturbation vector approach usually boils down to carefully placed equalities in the inequality tests done during intersection calculations. For certain tests, e.g., polygon-vs-polygon in 3D, the implementation may be non-trivial.

It should be noted that in most cases it is not disastrous to include a few extra voxels in the resulting voxelization in the rare case that a degenerate intersection occurs. A conservative intersection test that reports an intersection even when the input primitive and the intersection target merely touch each other can only add to the voxelization, and hence never break its connectivity or separability properties. Thinness may be compromised, and it depends on the application whether this is a concern or not. The principal value of the perturbation method is that it simplifies our analysis by removing the need to consider potential degeneracies.

4. Intersection targets in 2D

Let us first consider voxelization in 2D, i.e., rasterization. To follow the common nomenclature, we call our voxels pixels, and our input geometry may be zero-, one-, or two-dimensional in both primitive and effective dimensions. We assume that the effective dimension of the input is known in order to choose the appropriate voxelization strategy. We rely heavily on Jordan curve theorem that states that any continuous path that crosses from the inside to the outside of a simple closed curve must intersect this curve in at least one point. Also note that due to the handling of degenerate cases as described above, we do not need to separately consider cases where a path crosses through a point where multiple curves intersect or meet, as any such situation is resolved by the infinitesimal perturbation of the crossing path.

4.1. One-dimensional input

Input that is effectively one-dimensional may consist of either truly one-dimensional line segments or curves, or slim two-dimensional primitives. When voxelizing into a pixel grid, we may require the result to be either 4-connected

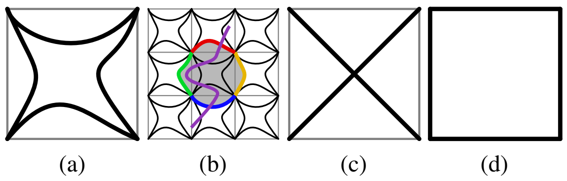


Figure 5: *Intersection target for 4-connected, 8-separating voxelization of one-dimensional input in 2D. (a) The general shape where adjacent pixel corners are connected by continuous curves. (b) Input geometry that intersects the target in center voxel cannot exit the region shaded in gray except through its bounding curve, which consists of parts of the intersection targets of the center pixel's 4-neighbors. (c) By stretching the corner-connecting curves to meet at center, we obtain a practical cross-diagonal intersection target. (d) Another option is to push the curves to the boundary of the pixel.*

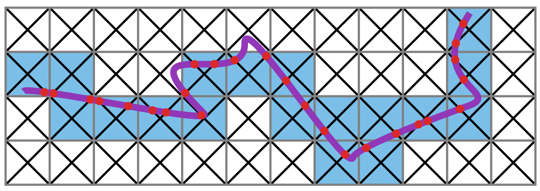


Figure 6: *An example voxelization of one-dimensional input with the cross-diagonal intersection target. Dots indicate intersections between the input curve and the intersection targets. 4-connectivity and 8-separability is achieved with a subset of a cover.*

or 8-connected, which are equivalent to 8-separability and 4-separability, respectively [COK95]. A trivial way to enforce 4-separability and hence 8-connectivity is to produce a cover of the input, but as our input is one-dimensional, we should expect a one-dimensional intersection target to suffice, and to produce fewer voxels.

4.1.1. 4-connected, 8-separating voxelization

Let us consider the intersection target shown in Figure 5a placed in every pixel. The important feature of this structure is that continuous curves connect every pair of adjacent corners of a pixel. Continuous input geometry that intersects the target in a pixel can extend to other pixels only by first passing through the intersection targets of this pixel's 4-neighbors, because parts of these form a simple closed curve around the pixel in question (Figure 5b). Therefore, the resulting voxelization of a continuous input path is 4-connected and as such 8-separating. An alternative way is to prove the 8-separability directly as done later in Section 5.2.1, from which the 4-connectivity follows.

Figure 6 shows an example of 4-connected voxelization

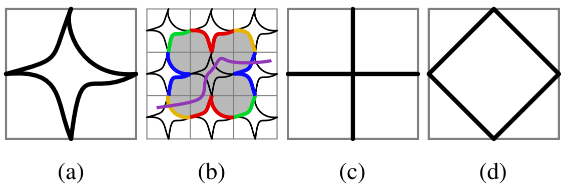


Figure 7: *Intersection target for 8-connected, 4-separating voxelization of one-dimensional input in 2D. (a) The general shape where adjacent edge centers are connected by continuous curves. (b) Input geometry that intersects the target in center voxel can can extend only to the pixel's 8-neighbors because the bounding curve of the gray region consists of their intersection targets. (c) Connecting the curves at center yields a practical crosshairs intersection target. (d) Connecting adjacent edge centers with straight line segments yields the diamond pattern used in common graphics APIs for line rendering.*

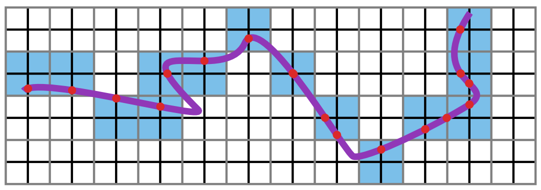


Figure 8: *An example 8-connected and 4-separating voxelization of one-dimensional input with the crosshairs intersection target.*

of one-dimensional input using the cross-diagonal target of Figure 5c, where the result contains fewer voxels than the two-dimensional full-pixel target would produce. The square target of Figure 5d is interesting only due to its ability to ignore point-like input with effective dimension of zero, including any input that falls in entirely inside a single pixel. Otherwise it produces the same result as the full-pixel target.

4.1.2. 8-connected, 4-separating voxelization

By modifying the previous intersection target, we can easily produce 8-connected, 4-separating voxelization of one-dimensional input. Figure 7a illustrates a general intersection target where midpoints of adjacent edges are connected by continuous curves. As illustrated in Figure 7b, we again obtain a simple closed curve around a pixel whose intersection target the input intersects. This curve consists of parts of the intersection targets of the pixel's 8-neighbors, and therefore prevents continuous input geometry from escaping without intersecting at least one of these. Again, we could show the 4-separability directly using the proof in Section 5.2.1, from which 8-connectivity follows

A practical intersection target is obtained by connecting the midpoints of pixel edges to the pixel center (Figure 7c).

This is a very simple target to intersect against, even for curved input primitives. OpenGL and DirectX graphics APIs employ a "diamond exit" rule for line rendering, which corresponds to the intersection target in Figure 7d, obtained as a special case of the general shape. The diamond target is more eager to include a pixel in the result than the crosshairs target—consider the input protruding into the pixel from the corner while remaining in one quadrant. This may yield a more aesthetically pleasing result at the cost of producing pixels that are unnecessary for 8-connectivity.

Thinness. In 8-connected voxelization, we may additionally be concerned about the so-called *thinness* of the result, loosely defined as producing a layer only one voxel thick in the direction of the major axis of the normal vector of surface or curve. Let us consider a parameterized continuous and differentiable one-dimensional input curve $c = (f_x(t), f_y(t)), t \in [0,1]$ where $|\mathrm{d}f_y/\mathrm{d}f_x| \geq 1$ for all t. Because c is always oriented closer to y axis than x axis, we would like to produce at most one voxel for every horizontal layer in S^d.

It is easy to see that the crosshairs target of Figure 7c fulfills this goal. Consider c intersecting the horizontal line segment at voxel V at point (x_0, y_0), where $y_0 = Y + \frac{1}{2}$, and $Y \in \mathbb{Z}$. Within this horizontal pixel layer, $y \in [y_0 - \frac{1}{2}, y_0 + \frac{1}{2}]$, and we have the bound $x \in [x_0 - \frac{1}{2}, x_0 + \frac{1}{2}]$ due to the limit on the ratio of absolute derivatives. Therefore, the only vertical line segment of an intersection target that c may potentially intersect must belong to the same crosshairs target whose horizontal line segment c intersects, as the x, y bounds contain no other such segment. Obviously, no other horizontal line segment on the layer can be intersected either. As such, no other voxels on the same layer will be in S^d. Using similar reasoning, the diamond target of Figure 7d can be seen to produce a thin voxelization.

Figure 8 shows an example of 8-connected voxelization of one-dimensional input using the crosshairs target of Figure 7c. The voxelization is 4-tunnel-free [COK95], as shown later in Section 5.2.2.

5. Intersection targets in 3D

In 3D, the relationship between connectivity and separability is less obvious than in 2D. Furthermore, with one-dimensional input, no sensible notion of separability can be given, but even then we may be interested in establishing various connectivity properties for the resulting voxelization.

The kind of reasoning used in Section 4 still applies for establishing connectivity properties for one-dimensional input, but the separability for two-dimensional input needs to be shown using a different strategy. We will again employ the Jordan curve theorem—or, more appropriately, the Jordan-Brouwer separation theorem which is the 3D analog.

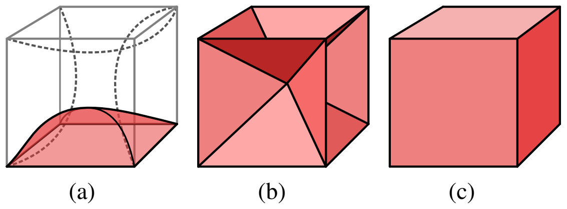


Figure 9: *Intersection target for 6-connected voxelization of one-dimensional input in 3D. (a) In the general shape, each face is closed by a surface within the voxel. Only the piece for the bottom face is shown here for clarity. (b) One possible realization is closing each face with pyramids that meet at the center. This corresponds to octahedral tessellation of space, where the centers of the octahedra are located at centers of voxel faces. (c) Another option is to push the surfaces to the boundaries of the voxels. Note that the intersection target is still two-dimensional, consisting of only the surface of the cube. Obviously, this corresponds to a cubical tessellation of space.*

This time, the simple, closed region is formed by the two-dimensional input geometry, and the paths between inside and outside regions are formed by piecing together parts of the intersection targets.

5.1. One-dimensional input

Effectively one-dimensional input in 3D may consist of one-, two-, or three-dimensional primitives. Even though such input cannot produce separation in 3D space, we may require the voxelization of the input to be 6-connected or 26-connected to allow, for example, flow simulation or a fill to proceed within the voxelized curve(s), or simply to guarantee that the appearance of a visualized voxelization is not missing pieces. We shall derive intersection targets for both connectivities below.

5.1.1. 6-connected voxelization

Following the 2D analogue from Section 4.1.1, let us consider the 3D intersection target illustrated in Figure 9a. For each face of the voxel, we construct a continuous surface that prevents curves from outside the voxel to enter the voxel and exit through some other face without intersecting this surface. Following the exact same reasoning as in the 2D case, we can see that a continuous one-dimensional input object cannot extend between two voxels without forming a 6-connected path of voxels in between.

Two practical realizations of this general shape are illustrated in Figure 9b, c, corresponding to octahedral and cubical tessellation of space, respectively. The practical relevance of these is dubious, as we may expect the voxelization to contain almost as many voxels as a cover produced by intersecting the input against the entire 3D voxel. In some

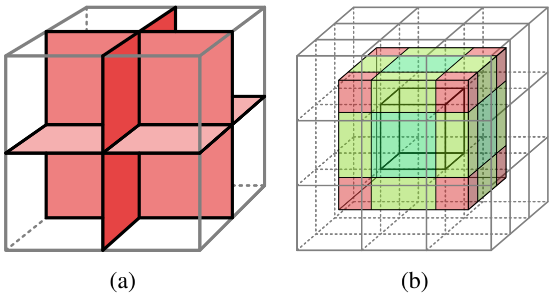


Figure 10: *Intersection target for 26-connected voxelization of one-dimensional input in 3D. (a) The target consists of three quadrilaterals that bisect the voxel along all three axes. This subdivides the space into cubes centered at voxel corners. (b) Input geometry that intersects the target in the center voxel cannot escape without intersecting the target of a 26-neighbor, because parts of those form a closed cube around the voxel with twice the side length.*

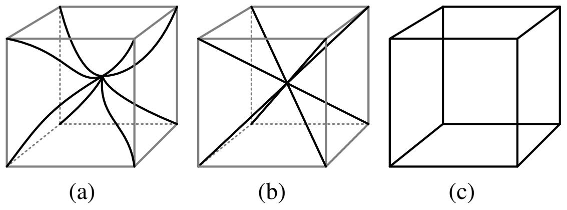


Figure 11: *Intersection target for 26-separating voxelization of two-dimensional input in 3D. (a) In the most general case, it is enough for the intersection target to connect every corner of the voxel to each other in one way or another. Here they connect to a common point inside the voxel, but that is not necessary. (b) In practice, straight line segments on space diagonals produces a simple, symmetrical target. (c) In some situations it may be more convenient to use a target that connects the corners along the edges of the voxel, as the individual line segment vs. surface intersection tests may be shared among neighboring voxels.*

situations it may be easier to intersect the input against an intersection target composed of 2D sheets than the 3D voxel, but in general these intersection targets are probably useful only when it is needed to keep the number of resulting voxels as small as possible.

5.1.2. 26-connected voxelization

The case of 26-connected voxelization of effectively one-dimensional input is more interesting. Producing a cover for this much more relaxed connectivity requirement would be clearly overkill, including many unnecessary voxels. The existing algorithms are restricted to input that is formed out of one-dimensional primitives, e.g., line segments or parameterized curves. In contrast, our method is able to construct a 26-connected voxelization out of input that is only effectively one-dimensional but composed of two- or three-dimensional primitives.

Figure 10a shows an intersection target suitable for this purpose. Connectivity follows from a similar argument that was used in Section 4.1.2 for 8-connected voxelization in 2D. Let us assume that the input intersects the target in voxel V. Parts of the intersection targets of the 26-neighbors of V form a box with side twice as long as a single voxel (Figure 10b). This is a simple, closed surface, and if the input geometry contains a continuous path from V to voxels outside the immediate neighbors, it must intersect this enclosing box and therefore intersect the target in $N_{26}(V)$.

It should be noted that the result is not thin in the same sense as in the 2D case with the crosshairs target. Therefore, an algorithm that steps along a curve in the dominant direction and outputs exactly one voxel per layer may produce a 26-connected voxelization with fewer voxels. The main benefit of our method is that it works for input other than one-dimensional primitives.

5.2. Two-dimensional input

Arguably the most relevant use for 3D voxelization is the discretization of two-dimensional input. In this case, the separability of the resulting voxelization is of main interest. Considering, e.g., casting a ray against the voxelized surface, we may opt to have our rays form 26-connected paths in order to test as few voxels as possible, in which case the voxelization of the surfaces has to be 26-separating to guarantee that the discretized ray does not penetrate an originally separating surface. Alternatively, if a ray march tests a 6-connected path of voxels, a sparser 6-separating voxelization of the input is sufficient. Cohen-Or and Kaufman [COK97] provide an in-depth analysis.

5.2.1. 26-separating voxelization

A cover, i.e., using the entire 3D voxel as the intersection target, is 26-separating as shown by Cohen-Or and Kaufman [COK95]. However, because our input is effectively two-dimensional, intersection targets composed of one-dimensional pieces should be sufficient to guarantee finding the relevant voxels.

Let us consider the intersection target in Figure 11a. We claim that the only property necessary to guarantee 26-separation is that the intersection target connects the corners of the voxel to each other. This is not trivial to see, so we will formulate an indirect proof using the Jordan curve theorem. It is important to keep in mind that degeneracies are handled by the infinitesimal perturbation method described in Section 3.2, and we therefore need not worry about cases where the surface, e.g., passes diagonally through the voxel corner. In other words, the perturbation guarantees that even in this case, intersection between an intersection target and the surface happens inside one of the voxels, and never exactly at the corner, edge, or face. If the input is composed of three-dimensional, volumetric primitives, but still forms an effectively two-dimensional surface (e.g., a spherical shell with nonzero thickness), we shall consider a suitable surface contained within this volume.

Restating our assumptions, let us have a simple, closed two-dimensional input surface S embedded in $\mathbb{R}^3$, separating it into two nonempty sets I and O. Assume that S^d is a voxelization of S produced using the above general intersection target, and additionally that there are nonempty sets I^d and O^d consisting of voxels whose entire 3D volume belong in I and O, respectively.

To show that S^d is 26-separating, we must show that there exists no path $\Pi = (V_0, \ldots, V_n)$ with $V_{i+1} \in N_{26}(V_i)$, $V_0 \in I^d$, $V_n \in O^d$, and $\Pi \cap S^d = \emptyset$. According to the Jordan curve theorem, every continuous path between a point in I and a point in O must intersect S. If the continuous path is entirely contained within the voxels in Π, the intersection happens in one of the voxels V_i. It is thus sufficient to construct one such continuous path and show that V_i must be included in S^d.

To construct such continuous path $C(\Pi)$, we connect pieces of intersection targets in each V_i. Because the intersection targets connect the corners of every voxel, we can indeed contain $C(\Pi)$ to lie within voxels present in Π, as moving to any of the N_{26} is possible along the intersection targets. We can start $C(\Pi)$ at any point in the intersection target of V_0, as $V_0 \in I^d$ and hence any point within V_0 is in I. Similarly, the endpoint of $C(\Pi)$ lying in V_n is trivially in O.

As $C(\Pi)$ is a simple, continuous curve between a point in I and a point in O, it must intersect S at some point $p \in \mathbb{R}^3$. Because $C(\Pi)$ is entirely contained within the volume corresponding to voxels in Π, the intersection point p must also lie in one of those voxels V_i which is part of Π. This now contradicts our definition of Π, because p is both on surface S and on the intersection target of V_i. Therefore, $V_i \in S^d$, and a 26-connected path Π as defined above cannot exist, implying that S^d is 26-separating.

The above reasoning holds for any intersection target that allows constructing a continuous path $C(\Pi)$ that is contained within the volume covered by voxels in 26-connected Π. This gives a lot of flexibility in formulating practical intersection targets. Figure 11b, c shows two options, but many others exist, especially if we do not care about symmetry.

5.2.2. 6-separating voxelization

A 6-separating voxelization may have many fewer voxels than a 26-separating one, and it is therefore generally preferable in applications where 26-separability is not required. The intersection target illustrated in Figure 12a connects the centers of voxel faces to each other, and can be seen to produce a 6-separating voxelization. The argument is exactly the same as in the previous case, except that now the hypothetical separability-violating voxel path Π is 6-connected. Therefore, a continuous $C(\Pi)$ can be formed as long as it is possible to move between neighboring voxels in a 6-neighbor sense, i.e., through the faces, which the proposed intersection target clearly allows. The most practical symmetrical realization of the general shape is probably the crosshairs target in Figure 12b.

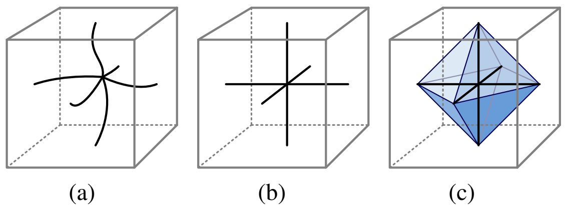


Figure 12: *Intersection target for 6-separating voxelization of two-dimensional input in 3D. (a) The general intersection target connects centers of faces to each other. (b) Straight line segments yield a particularly simple crosshairs shape. (c) 3D generalizations of the diamond rule (e.g., [SS10]) produce a 6-separating result because they correspond to a superset of the crosshairs target.*

It immediately follows that any sensible 3D extension of the diamond rule used in 2D produces a 6-separating voxelization due to inclusion of the crosshairs target. For example, the method of Schwarz and Seidel [SS10] always includes any voxel where an input triangle intersects the octahedron spanned between voxel's face centers (Figure 12c) and is therefore 6-separating. However, using an unnecessarily large target will include voxels that are not required for 6-separation, indicating that their method cannot be minimal.

A simple analysis also reveals that the GPU-based voxelization method used by Forest et al. [FBP09] and Crassin et al. [CNS*11] is 6-separating. They do three passes over the input, and in each pass rasterize the primitives along three axis-aligned projections. For each resulting pixel, the depth of the primitive is evaluated at the center, and the voxel whose center is closest in depth is included in S^d. It is easy to see that this is equivalent to using the crosshairs intersection target of Figure 12b, where each rasterization pass corresponds to intersecting the input against the crosshair segments oriented along the projection axis. By extension, this rasterization-based method would produce 6-separating result even for curved primitives, while requiring only point evaluations of depth. This property might be quite difficult to ascertain without the techniques used above.

Thinness. The voxelization produced by the crosshairs intersection target is also thin in the following sense. Consider a possibly curved, continuous and everywhere differentiable surface primitive where, at every point, the z component of the normal vector has the largest absolute value. For thinness, we require that no more than one voxel is produced in each z-oriented column of voxels. Let the surface intersect the z-oriented crosshair segment of voxel V at point (x_0, y_0, z_0), where $x_0 = X + \frac{1}{2}, y_0 = Y + \frac{1}{2}$, and $X, Y \in \mathbb{Z}$. In this (z-oriented) voxel column, $x \in [x_0 - \frac{1}{2}, x_0 + \frac{1}{2}]$, and similarly for y. Consider slicing the surface with an xz plane at $y = y_0$. Because of the normal vector leaning towards z at all points, we have $|\mathrm{d}z/\mathrm{d}x| \leq 1$. Hence, z also has bounds $z \in [z_0 - \frac{1}{2}, z_0 + \frac{1}{2}]$ in this x range when keeping $y = y_0$, and the surface thus cannot reach the x-oriented crosshair segment in voxels other than V in this column. The same holds for the y-oriented crosshair segments. Note that in general the surface z may reach the middle of another voxel in the column, but this cannot happen on the xz and yz planes where the crosshair segments lie. Furthermore, the surface obviously cannot intersect any other z-oriented crosshair segments in the same column. Hence, no other voxels of this column besides V can be included in the result, from which thinness follows.

Voxelization with a single 2D projection. An interesting corollary follows from the thinness property and the equivalence of the 3-axis rasterization [FBP09, CNS*11] with the crosshairs target. When a planar triangle is projected along the dominant axis of its geometric normal vector, every pixel whose center lies in the projected triangle produces a unique voxel in the corresponding voxel column. The other projection directions cannot produce any more voxels in these columns, as the voxelization with the crosshairs target is thin. For all such pixels, it is therefore enough to consider projection along the dominant axis alone, and only the edges need further consideration. This suggests the following algorithm.

Rasterize the triangle projected into 2D along the dominant axis of normal. For pixels where the projected triangle contains the pixel center, output the voxel whose center depth is closest to the depth of the triangle evaluated at pixel center, as in previous methods [FBP09, CNS*11]. For pixels that contain an edge of the triangle but where the triangle does not overlap the center, find the extents of the triangle on a horizontal line bisecting the pixel, clamp these to the horizontal pixel extents, and evaluate depth of triangle at these two points. If the values are on different sides of a voxel's center depth, output the corresponding voxel. If not, repeat the test for the vertical direction. The first test (triangle vs. pixel center) corresponds to intersecting the triangle against the crosshair segment parallel to the projection axis, and the latter tests correspond to testing the triangle against the two crosshair segments perpendicular to the projection axis. This way, the 6-separating voxelization can be produced based on just one 2D projection.

Tunnel-freeness. Following the definition of Cohen-Or and Kaufman [COK95], a 6-tunnel-free voxelization is such that any line segment that connects the centers of two 6-neighboring voxels outside S^d does not intersect S. It is easy to see that using the crosshairs target in Figure 12b fulfills this requirement, as any such segment that intersects S will produce at least one of the neighboring voxels in S^d. The same holds for 2D voxelization using the crosshairs target in Figure 7c. We could similarly obtain 8-tunnel-free voxelization in 2D by using an intersection target that connects the pixel center to its corners and edge midpoints, 18-tunnel-free voxelization in 3D with a target that connects voxel center to face centers and edge midpoints, and 26-tunnel-free voxelization with a target where voxel center is connected to face centers, edge midpoints, and corners.

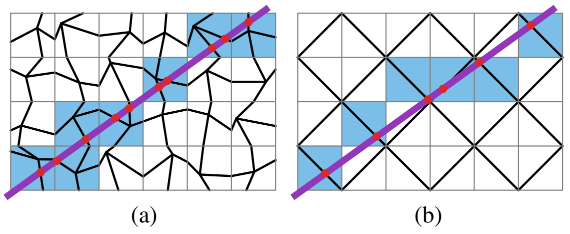


Figure 13: *Modifications to the crosshairs target for one-dimensional input in 2D. (a) Randomizing the connection points within pixels and on pixel edges produces an irregular pattern that is still 4-separating and hence 8-connected. (b) Pushing the edge connection points all the way to pixel corners yields a pattern where the intersection target in each pixel consists of a single diagonal line segment.*

6. Extensions

So far we have considered only the situations where the intersection target is the same in every pixel or voxel. This is not strictly required, as none of our connectivity and separability arguments relies directly on this property.

While any of the proposed intersection targets can be altered in various ways, we shall here consider only the 4-separating crosshairs target for one-dimensional input in 2D (Figure 7c), and the 6-separating crosshairs target for two-dimensional input in 3D (Figure 12b), as they are the easiest to manipulate.

Figure 13a illustrates a variant of the former intersection target where the connections are made between random points on pixel edges to a random point within the pixel, and the resulting voxelization. This "foam" retains the connectivity and separability properties of the voxelization, but has a much less regular structure. This may improve, e.g., the spectral properties of the resulting voxelization, if that is of interest in a given application.

An even more drastic modification to the intersection target is to push the connecting segments to meet at pixel corners, as illustrated in Figure 13b. It is easy to see that

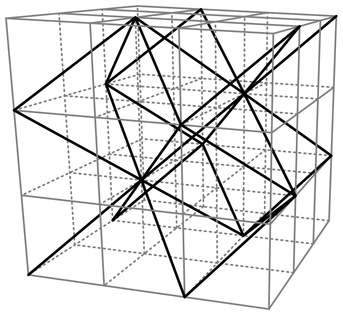


Figure 14: *Modification of the crosshairs target for 3D voxelization of two-dimensional input, obtained by moving the face connection points to voxel corners. This merges the six "arms" of the crosshairs into two collinear segments, yielding a pattern where the intersection target in each voxel consists of a single space diagonal. Despite the sparsity, this set of intersection targets produces a 6-separating voxelization.*

4-separability holds when shown using a 2D analog of the separability proof in Section 5.2.1. Albeit diverging considerably from the simple crosshairs target, this alternating diagonal target still allows a continuous 4-connected $C(\Pi)$ to be formed using only the voxels in Π, from which 4-separability and 8-connectivity follows. The resulting voxelization is somewhat clunky, but it is still a subset of a cover of the input, and hence never arbitrarily far from the input geometry.

Similar modifications can be carried out to the crosshairs target for voxelizing two-dimensional input in 3D (Figure 12b). Randomization of the lattice works as in the 2D case, and by pushing the connection vertices to voxel corners, we obtain a strikingly simple intersection target consisting of a single space diagonal in each voxel, as illustrated in Figure 14. The four possible space diagonals must be alternated as shown to maintain connectivity between voxels, and the pattern repeats with a period of 2^3 voxels. The resulting voxelization exhibits similar clunkiness as in the 2D case, but it is nonetheless a 6-separating subset of a cover and hence reasonably faithful to the input geometry.

In applications where the simplicity of the intersection target and the small number of resulting voxels are of higher importance than aeshetic aspects—e.g., if the voxelization is not to be visualized directly—using the alternating single-diagonal targets may be the most efficient strategy.

7. Conclusions

We have presented a general voxelization scheme based on intersection targets, along with a number of targets that are suitable for 2D and 3D voxelization of input geometry with various effective dimensions. By choosing appropriate intersection targets, the resulting voxelization can be easily shown to have various topological connectivity and separability properties. Unlike most previous methods, our scheme is applicable to curved input primitives in addition to flat ones, and trivially independent of input tessellation.

References

[CNS*11] Crassin C., Neyret F., Sainz M., Green S., Eisemann E.: Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum (Proceedings of Pacific Graphics 2011) 30*, 7 (2011).

[COK95] Cohen-Or D., Kaufman A.: Fundamentals of surface voxelization. *Graph. Models Image Process. 57*, 6 (1995), 453–461.

[COK97] Cohen-Or D., Kaufman A.: 3D line voxelization and connectivity control. *Computer Graphics and Applications, IEEE 17*, 6 (1997), 80–87.

[DCB*04] Dong Z., Chen W., Bao H., Zhang H., Peng Q.: Real-time voxelization for complex polygonal models. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (2004), pp. 43–50.

[ED06] Eisemann E., Décoret X.: Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), pp. 71–78.

[FBP09] Forest V., Barthe L., Paulin M.: Real-time hierarchical binary-scene voxelization. *Journal of Graphics Tools 14*, 3 (2009), 21–34.

[FC00] Fang S., Chen H.: Hardware accelerated voxelization. *Computers and Graphics 24*, 3 (2000), 433–442.

[HYFK98] Huang J., Yagel R., Filippov V., Kurzion Y.: An accurate method for voxelizing polygon meshes. In *Proceedings of the 1998 IEEE symposium on Volume visualization* (1998), pp. 119–126.

[KS87] Kaufman A., Shimony E.: 3D scan-conversion algorithms for voxel-based graphics. In *Proceedings of the 1986 workshop on Interactive 3D graphics* (1987), pp. 45–75.

[Pan11] Pantaleoni J.: VoxelPipe: a programmable pipeline for 3D voxelization. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (2011), pp. 99–106.

[SS10] Schwarz M., Seidel H.-P.: Fast parallel surface and solid voxelization on GPUs. *ACM Trans. Graph. 29*, 6 (2010), 179:1–179:10.

[VKK*03] Varadhan G., Krishnan S., Kim Y. J., Diggavi S., Manocha D.: Efficient max-norm distance computation and reliable voxelization. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), SGP '03, pp. 116–126.

[WK93] Wang S. W., Kaufman A. E.: Volume sampled voxelization of geometric primitives. In *Proceedings of the 4th conference on Visualization '93* (1993), pp. 78–84.

[WME03] Widjaya H., Möller T., Entezari A.: Voxelization in common sampling lattices. In *Proceedings of Pacific Graphics 2003* (2003), pp. 497–501.

[ZCEP07] Zhang L., Chen W., Ebert D. S., Peng Q.: Conservative voxelization. *Vis. Comput. 23*, 9 (2007), 783–792.