Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware

Kenneth E. Hoff III, Tim Culver, John Keyser, Ming Lin, Dinesh Manocha

University of North Carolina at Chapel Hill
Department of Computer Science

Abstract: We present a new approach for computing generalized 2D and 3D Voronoi diagrams using interpolation-based polygon rasterization hardware. We compute a discrete Voronoi diagram by rendering a three dimensional distance mesh for each Voronoi site. The polygonal mesh is a bounded-error approximation of a (possibly) non-linear function of the distance between a site and a 2D planar grid of sample points. For each sample point, we compute the closest site and the distance to that site using polygon scan-conversion and the Z-buffer depth comparison. We construct distance meshes for points, line segments, polygons, polyhedra, curves, and curved surfaces in 2D and 3D. We generalize to weighted and farthest-site Voronoi diagrams, and present efficient techniques for computing the Voronoi boundaries, Voronoi neighbors, and the Delaunay triangulation of points. We also show how to adaptively refine the solution through a simple windowing operation. The algorithm has been implemented on SGI workstations and PCs using OpenGL, and applied to complex datasets. We demonstrate the application of our algorithm to fast motion planning in static and dynamic environments, selection in complex user-interfaces, and creation of dynamic mosaic effects.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.3 [Computer Graphics]: Picture/Image Generation.

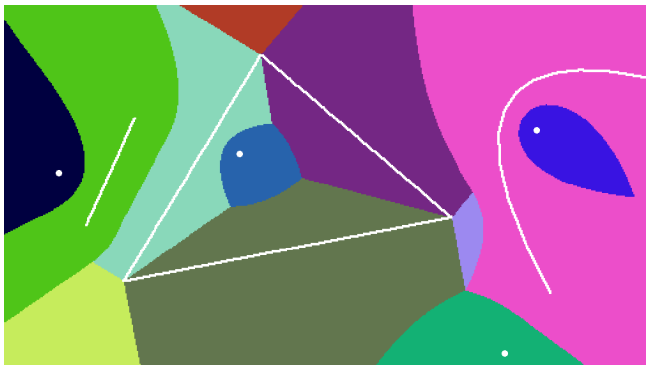
Additional Key Words: Voronoi diagrams, graphics hardware, polygon rasterization, interpolation, motion planning, proximity query, medial axis, OpenGL, framebuffer techniques.

1 INTRODUCTION

Given a set of primitives, called Voronoi sites, a Voronoi diagram partitions space into regions, where each region consists of all points that are closer to one site than to any other. Voronoi diagrams have been used in a number of applications including visualization of medical datasets, proximity queries, spatial data manipulation, shape analysis, computer animation, robot motion planning, modeling spatial structures and processes, pattern recognition, and locational optimization. The concept of Voronoi diagrams has been around for at least four centuries, and since the 1970s, algorithms for computing Voronoi diagrams of geometric primitives have been developed in computational geometry and related areas.

e-mail: {hoff,culver,keyser,lin,dm}@cs.unc.edu
WWW: http://www.cs.unc.edu/~geom/voronoi/

SIGGRAPH 99, Los Angeles, CA USA



Cover Plate: Discrete approximation of the generalized Voronoi diagram of four points, a line, a triangle, and one cubic Bézier curve computed interactively on a PC.

Good theoretical and practical algorithms are known for computing ordinary Voronoi diagrams of points in any dimension. Ordinary Voronoi diagrams can be generalized in many different ways by using different distance functions and site shapes. A common generalization is to compute the diagram for higher-order sites, such as lines and curves. This greatly increases the complexity since the boundaries of the diagram are composed of high-degree algebraic curves and surfaces, and their intersections; the boundaries of an ordinary point Voronoi diagram are linear. No practically efficient and numerically robust algorithms are known for constructing a topologically consistent, continuous representation of generalized Voronoi diagrams.

Given the practical complexity of computing an exact generalized Voronoi diagram, many authors have proposed approximate algorithms. Interesting approaches include computing the Voronoi diagram of a point-sampling of the sites, adaptively subdividing space to locate the Voronoi boundary, and point-sampling the space to form a volumetric representation of the diagram. In practice, these previous algorithms take considerable time and memory on large numbers of input sites, or are restricted in generality.

Main Contributions: In this paper, we present an approach that computes discrete approximations of generalized Voronoi diagrams to an arbitrary resolution using polygon rasterization hardware. Our contributions include:

1. Efficient methods to approximate the distance function, with bounded error, for points, lines, polygons, polyhedra, curves, and curved surfaces using a polygonal mesh that is linearly interpolated by graphics hardware.
2. Efficient algorithms to find Voronoi boundaries and neighbors, and to construct Delaunay triangulations.

3. Techniques to construct weighted and farthest-site generalized Voronoi diagrams in 2D and 3D.

4. Demonstration of the effectiveness of our approach to the following applications:

   - Fast motion planning in static and dynamic environments
   - Selection in complex user-interfaces
   - Generation of dynamic mosaics

The resulting techniques have been effectively implemented on PCs and high-end SGI workstations using the OpenGL graphics library. A 2D example computed in real-time is shown in the cover plate. Our techniques improve upon the state of the art in following ways:

- **Generality**: We make no assumption with respect to input primitives. We only need to mesh the distance function of a site over a grid of point samples.
- **Efficiency**: We show that our approach is quite fast. Its speed arises from using coarse polygonal approximations of the distance functions while still maintaining a specified error bound, using polygon rasterization hardware to reconstruct the distance values, and using the Z-buffer depth comparison to perform distance comparisons. We demonstrate the 2D approach on models composed of nearly 100K triangles in a real-time motion planning application through a complex dynamic scene. We derive efficient meshing strategies for polygonal models in 3D, and show the results of a prototype implementation that demonstrates its potential.
- **Tight Bounds on Accuracy**: Although our approach produces a discretized Voronoi diagram, all sources of error are enumerated and techniques are given to produce output within any specified tolerance.
- **Ease of Implementation**: The approach can be easily implemented on current graphics systems. The special cases are limited and the problem reduces to simply meshing a distance function for any new site.

2 RELATED WORK

The concept of Voronoi diagrams has been around for at least four centuries. In his treatment of cosmic fragmentation in *Le Monde de Mr. Descartes, ou Le Traite de la Lumière*, published in 1644, Descartes uses Voronoi-like diagrams to show the disposition of matter in the solar system and its environment. The first presentations of this concept appeared in the work of [Diric50] and [Voron08]. Algorithms for computing Voronoi diagrams have been appearing since the 1970s. See the surveys by [Auren91] and [Okabe92] on various algorithms, applications, and generalizations of Voronoi diagrams.

2.1 Voronoi Diagrams of Points

Among the algorithms known for computing Voronoi diagrams of points in 2D, 3D, and higher dimensions are the divide-and-conquer algorithm proposed by [Shamo75] and Fortune's sweepline algorithm [Fortu86]. Numerically robust algorithms for constructing topologically consistent Voronoi diagrams have been proposed by [Inaga92, Sugih94]. A number of implementations in exact and floating-point arithmetic are also available.

2.2 Generalized Voronoi Diagrams

Algorithms have been proposed for constructing Voronoi diagrams of higher order sites. Two broad approaches based on incremental and divide-and-conquer techniques have been summarized in [Okabe92]. The set of algorithms includes divide-and-conquer algorithms for polygons [Lee82, Held97], an incremental algorithm for polyhedra [Milen93b], and 3D tracing for polyhedral models [Milen93, Sherb95, Culve99]. Curved sites and CSG objects are handled in [Chian92, Dutta93, Hoffm94]. In all these cases, the computation of generalized Voronoi diagrams involves representing and manipulating high-degree algebraic curves and surfaces and their intersections. As a result, no efficient and numerically robust algorithms are known for computing them.

2.3 Approximate Voronoi Diagrams

Many authors compute approximations of generalized Voronoi diagrams based on the Voronoi diagram of a point-sampling of the sites [e.g. Sheeh95]. However, deriving any error bounds on the output of such an approach is difficult, and the overall complexity is not well understood.

[Vleug95] and [Vleug96] have presented an approach that adaptively subdivides space into regular cells and computes the Voronoi diagram up to a given precision. [Laven92] uses an octree representation of objects and performs spatial decomposition to compute the approximation. [Teich97] computes a polygonal approximation of Voronoi diagrams by subdividing the space into tetrahedral cells. All these algorithms take considerable time and memory for large models composed of tens of thousands of triangles, and cannot easily be extended to directly handle dynamic environments.

The idea of using polygon rasterizing hardware and rendering of cones to construct 2D Voronoi diagrams of points is suggested in [Haebe90] and in the OpenGL 1.1 Programming Guide [Woo97].

2.4 Graphics Hardware

Polygon rasterization graphics hardware has been used for a number of geometric computations, such as visualization of constructive solid geometry models [Rossi86, Goldf89] and interactive inspection of solids, including cross-sections and interferences [Rossi92]. Algorithms for real-time motion planning using raster graphics hardware have been proposed by [Lengy90].

3 OVERVIEW

In this section, we present the basic concepts important to our approach. We give a formal definition of generalized Voronoi diagrams and present a simple brute-force strategy for computing a discrete approximation. We then show how we may greatly accelerate this using graphics hardware.

3.1 Generalized Voronoi Diagrams

The set of input sites is denoted as $A_1, A_2, \ldots, A_k$. For any point p in the space, $dist(p, A_i)$ denotes the distance from the point p to the site A_i. The dominance region of A_i over A_j is defined by

$$Dom(A_i, A_j) = \{ p \mid dist(p, A_i) \leq dist(p, A_j) \}$$

For a site A_i, the Voronoi region for A_i is defined by

$$V(A_i) = \cap_{j \neq i} Dom(A_i, A_j)$$

The partition of space into $V(A_1)$, $V(A_2)$, …, $V(A_k)$ is called the *generalized Voronoi diagram*. The (ordinary) Voronoi diagram corresponds to the case when each A_i is an individual point. The boundaries of the regions $V(A_i)$ are called *Voronoi boundaries*. For primitives such as points, lines, polygons, and splines, the Voronoi boundaries are portions of algebraic curves or surfaces.

3.2 Discrete Voronoi Diagrams

Perhaps the simplest way to compute a discrete Voronoi diagram is to uniformly point-sample the space containing Voronoi sites. For each sample point, we find the closest site and its distance. Associating each point in space with its closest sample point induces a uniform subdivision into rectangular cells. For any point, we know the distance to the closest site to within the maximum distance between a point in space and a sample point, i.e. half the diagonal length of a cell.

A simple brute-force approach to find the closest sites is to iterate through all sample points, computing distances to all sites and recording the closest site and distance. The algorithm can be rearranged to iterate through the sites: for each site, compute distances to all sample points and update the current closest site and distance. The second arrangement is amenable to an implementation in graphics hardware.

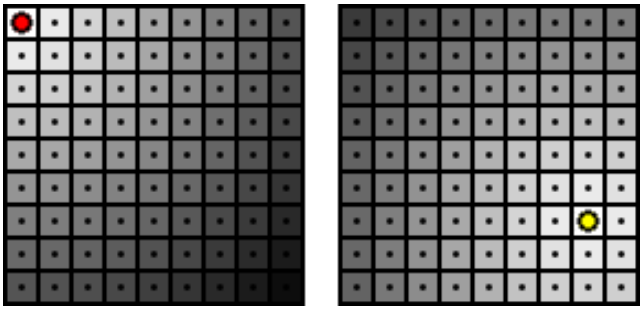


Figure 1: Image of the sampled distance functions for two point sites. Uniform point sampling induces a rectangular cell subdivision of space.

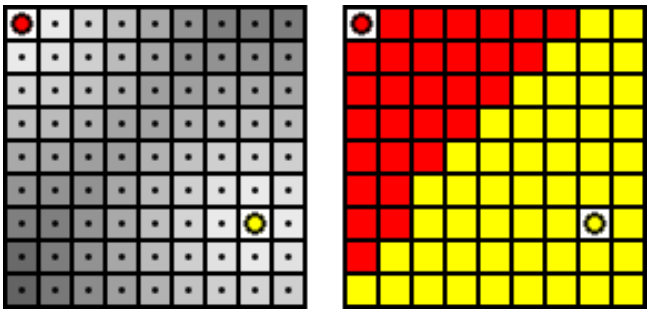


Figure 2: The two distance images are composited through a distance comparison operation. The current closest site and the distance to each site is updated based on the lesser distance value. The resulting Voronoi diagram is composed of a distance image (left) and an closest-site ID image (right).

3.3 Polygon Rasterization Hardware

Our approach makes use of standard Z-buffered raster graphics hardware for rendering polygons. The frame buffer stores the attributes (intensity or shade) of each pixel in the image space; the Z-buffer, or depth buffer, stores the *z*-coordinate, or depth, of every visible pixel. Given only the vertices of a triangle, the rasterization hardware uses linear interpolation to compute depth values across the triangle's surface. All raster samples covered by a triangle have an interpolated *z*-value.

3.4 Our Approach

A key concept for our approach is that of the *distance function* for a site, which gives, for any point, the distance to that site. The main idea of our approach is to render a polygonal mesh approximation to each site's distance function. Each site is assigned a unique color ID, and the corresponding distance mesh is rendered in that color using a parallel projection. We make use of two components of the graphics hardware: linear interpolation across polygons and the Z-buffer depth comparison operation. When rendering a polygonal distance mesh, the polygon rasterization reconstructs all distances across the mesh. The Z-buffer depth test compares the new depth value to the previously stored value. If the new value is less, the Z-buffer records the new distance, and the color buffer records the site's ID. In this way, each pixel in the frame buffer will have a color corresponding to the site to which it is closest, and the depth-buffer will have the distance to that site. In order to maintain an accurate Voronoi diagram, we bound the error of the mesh to be smaller than the distance between two sample points.

Our approach is inspired by an interesting sidenote in the OpenGL 1.1 Programming Guide [Woo97]. In the Section "Now That You Know" on "Dirichlet Domains", the authors briefly discuss a simple method to construct discretized 2D Voronoi diagrams for points using OpenGL graphics hardware. The authors mention the use of cones for Voronoi diagrams of points in 2D, but warn that the technique "might require thousands of polygons." We show that we can render cones using fewer than 100 triangles for a 1K×1K resolution grid and achieve the same level of accuracy. In addition, we generalize this approach to higher-order sites in both two and three dimensions.

4 THE DISTANCE FUNCTIONS

For both 2D and 3D, our discrete Voronoi diagram computation has been reduced to finding a 3D polygonal mesh approximation to the distance function of a Voronoi site over a planar 2D rectangular grid of point samples. The error in the approximation must be bounded so that by rendering this mesh using graphics hardware, we can efficiently and accurately compute the distances between the site and all of the point samples.

In this section, we describe the distance functions associated with various sites, and provide efficient methods for meshing these functions within a specified error tolerance.

4.1 2D Voronoi Diagrams

Denote the distance from a site A to each pixel location (x,y) by $dist(A,(x,y))$. The *distance function* of A is given by $d(x,y)=dist(A,(x,y))$. Meshing this function corresponds to approximating the graph of $d(x,y)$ with a polygonal model.

The three basic types of 2D sites are points, lines, and polygons. Their corresponding distance functions are shown in the table. In this section, we present algorithms for computing distance meshes for each of them.

2D site	Shape of Distance Function	Figure
Point	Right circular cone	3a
Line segment	"Tent"	3b
Polygon	Cones and tents	5

Table 1: Shape of Distance Functions for 2D Sites

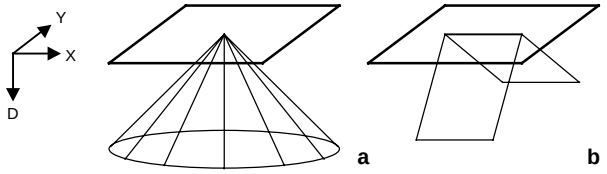


Figure 3: The distance meshes used for a point (left) and a line segment (right). The XY-plane containing the site is shown above each mesh.

4.1.1 Points in 2D

The distance function for a point in the plane is a right circular cone. We approximate cones as a triangle fan proceeding radially outward from the apex (Figures 3a and 4-left). A point's Voronoi region can potentially extend to any portion of the region of interest, and thus the radius at the cone's base must be of size $M\sqrt{2}$ if the scene is contained in an $M\times M$ square.The mesh's radial lines lie on the cone. The maximum error in distance occurs at the cone base between adjacent vertices. Because the cone is right circular, the error in approximating the circular base as viewed from above is equal to the error in distance.

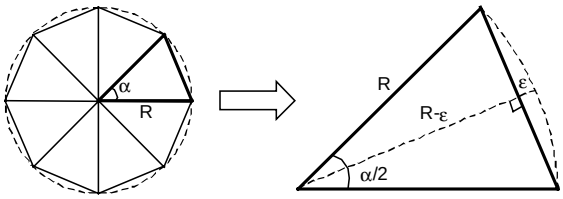


Figure 4: A single triangle of the meshed point distance function cone. α is the angle we wish to maximize, R is the radius of the cone (max dist between site and sample pt), and ε is the max error.

From this formulation (see Figure 4), we compute the maximum angle as:

$$\cos(\alpha/2) = \frac{R-\varepsilon}{R} \;\rightarrow\; \alpha = 2\cos^{-1}\left(\frac{R-\varepsilon}{R}\right)$$

For example, for a maximum distance error of no more than one pixel's width, a cone mesh for a 512×512 grid will require only 60 triangles. A 1024×1024 grid will require 85 triangles.

4.1.2 Line Segments in 2D

The distance function for a line segment is composed of three parts: one for the segment itself and one for each endpoint. The endpoints are treated the same way as points. The distance function for the line segment (excluding the endpoints) is just a "tent" (Figure 3b); its distance mesh is composed of two quadrilaterals. These represent the distance function exactly, so there is no error in the distance mesh representation. The only error for the line segment is in the cone mesh for the endpoint distance functions, as described in the previous section.

4.1.3 Polygons and Per-feature Voronoi Diagrams

It is often useful to consider sites as a collection of features, rather than as a single entity. For example, a line segment would be considered as three features: the two endpoints and the linear edge between them. By rendering the distance meshes for different features in different colors, we obtain a discrete approximation of a *per-feature Voronoi diagram*. Such diagrams are useful in several contexts: for example, the computation of a medial axis of a polygon. A picture of a per-feature Voronoi diagram for a polygon is given in Figure 5-left.

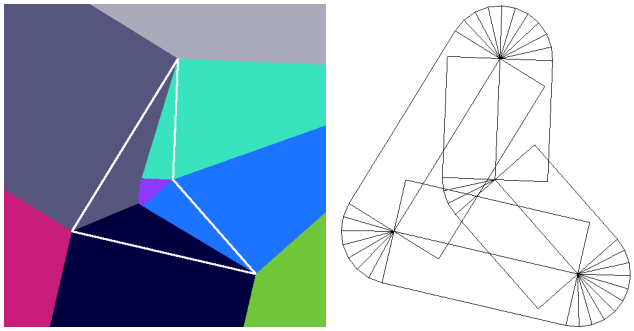


Figure 5: The per-feature Voronoi diagram of a quadrilateral (left). The corresponding distance mesh (right).

Polygons are rendered as a series of linear segments connected at the vertices. Each edge and vertex is a feature. For the vertices, rendering a triangle fan connecting two adjacent edges, rather than a full point distance mesh cone, saves on the total number of triangles computed and ensures that the distance meshes for adjacent features join smoothly. See Figure 5-right for an illustration.

4.2 3D Voronoi Diagrams

Our algorithm computes a 3D discrete Voronoi diagram slice-by-slice. Each slice is parallel to the (x,y)-plane and is computed independently.

Consider the slice $z=z_0$. To construct the intersection of the Voronoi diagram with this slice, consider the distance function for a site A, restricted to the slice. Denote the restricted distance function by $dist(x,y)=dist(A,(x,y,z_0))$. In this section, we describe $dist(x,y)$ for polygon, line segment, and point sites. As in the 2D case, computing the discrete Voronoi diagram is a matter of meshing the distance function $d=dist(x,y)$ for each site and rendering these meshes.

The distance meshes we give for the 3D problem are for a per-feature Voronoi diagram. Thus, a detached triangle site is treated as seven features: a polygon, three line segments, and three points. As in 2D per-feature diagrams, some features have a restricted region of influence.

3D site	Shape of distance function	Figure
Polygon	Plane	6
Line segment	Elliptical cone	7
Point	1 sheet of a hyperboloid of 2 sheets	8

Table 2: Shape of Distance Functions for 3D Sites

4.2.1 Polygons in 3D

The influence of this site in 3D is confined to the region formed by sweeping the polygon orthogonally through space, since points outside this region are considered to be closer to an edge or vertex of the polygon. In the slice, this region is a polygon, and $dist(x,y)$ is linear within this region, as illustrated in Figure 6. The distance to the site is computed at the vertices of the region, and a distance mesh composed of a single polygon is rendered. No meshing error

is incurred. If the polygon intersects the slice, the intersection is computed and the polygon is decomposed into two sub-polygons. Each sub-polygon is treated as above.

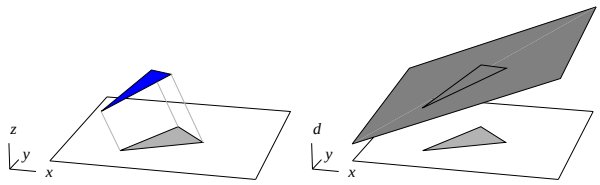


Figure 6: A polygonal site and its region of influence in a slice (left). The corresponding linear distance function (right).

4.2.2 Line Segments in 3D

The graph of the distance function for a line segment site is an elliptical cone (Figure 7). The apex of the cone lies at the intersection of the segment's line with the slice, and the cone's eccentricity is determined by the relative angle of the line and the slice. The 3D region of influence of a line segment lies between two parallel planes through the endpoints, since a point outside these planes is closer to one of the endpoints than to the segment.

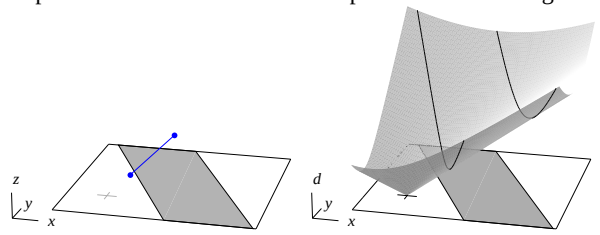


Figure 7: A line-segment site and its region of influence in a slice (left). The corresponding conical distance function (right).

4.2.3 Points in 3D

The distance function for a point site is shown in Figure 8. Its graph is one sheet of a hyperboloid of revolution of two sheets. If the point lies in the slice, the distance function is a cone rather than a hyperboloid. The region of influence for a single point is the entire slice.

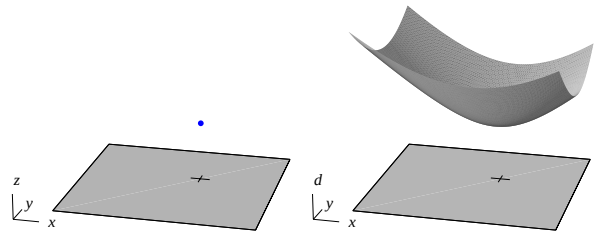


Figure 8: A point site and its region of influence in a slice (left). The corresponding hyperbolic distance function (right).

4.2.4 Meshes for Line Segments and Points in 3D

The construction of bounded-error meshes for the line-segment and point distance functions is detailed in [Hoff99]. The method attempts to minimize the complexity of the mesh by committing the maximum allowable error ε in each mesh cell. The structure of the mesh depends only on the *resolution* of the Voronoi diagram, defined by the ratio of the diameter M of the model to the maximum meshing error ε. The mesh structure is precomputed; during the Voronoi diagram construction, the mesh is constructed using table-lookup. Examples of the meshes produced by this method are shown in Figure 9.

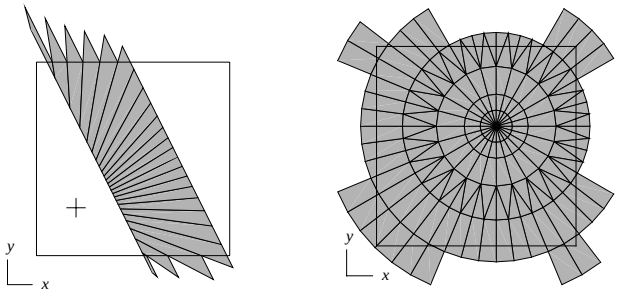


Figure 9: A bounded-error distance mesh for the line-segment site (left) and the point site (right).

4.3 Generalization to Curved Sites

The exact distance function for a curved site can be rather complicated, and for splines or algebraic curves is a high-degree algebraic function. We simplify this by creating a linear tessellation of the curved site, and then meshing the distance function of this approximation. We can use algorithms such as in [Filip87] and [Kumar96] to obtain bounded-error tessellations.

Figure 10 shows the mesh for a Bézier curve. Since the mesh for a linear segment is exact, the distance error for any of the linear segments is just the error in the deviation of the line from the original curve. The endpoints of the curve must be treated as points, just as for the line segment. The distance mesh for the "joints" between linear segments is a portion of the radial mesh of triangles. An overall maximum error bound of ε can be obtained for the entire curve by:

- tessellating the curve into linear segments with maximum error bound of ε;
- rendering the distance mesh for the linear segments; and
- treating the endpoints and joints as points, and rendering each point distance mesh with maximum error bound of ε.

This approach generalizes to 3D surfaces, which can be tesselated into a polygonal mesh. The error is bounded in a similar way.

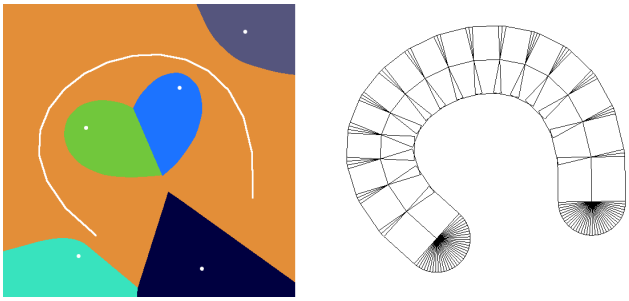


Figure 10: The Voronoi diagram of a Bézier curve and 5 points (left). The distance mesh for the Bézier curve that has been tessellated into 16 segments (right).

4.4 Weighted and Farthest-site Diagrams

In a *weighted* Voronoi diagram, the distance functions are additively or multiplicatively weighted [Okabe92]. Translation of a distance mesh along the distance axis accounts for additive

weights. Linear scaling along the distance axis accounts for multiplicative weights. In 2D, this is equivalent to changing the angle of the cone or tent. Scaling the distance mesh also scales the meshing error.

In a *farthest-site* Voronoi diagram, the farthest site from each point is found. Unlike in the nearest-site diagram, the distance function monotonically decreases as we move away from the site. We obtain the proper distance relationships by negating the distance functions. In practice, however, we need only reverse the depth-test (less-than to greater-than) and change the depth initialization from ∞ to 0.

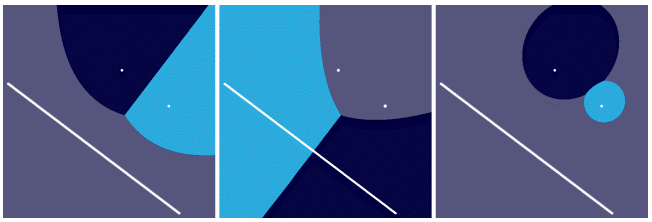


Figure 11: Standard nearest-site Voronoi regions (left). Farthest regions for the same sites (middle). Weighted regions (right). Weights: line, 2; dark point, 1; light point, 0.5.

5 BOUNDARIES AND NEIGHBORS

A continuous Voronoi diagram representation usually specifies the *Voronoi boundaries* that separate the set of Voronoi regions. In our discrete representation, we must search for the boundaries using approaches similar to iso-surface extraction and root-finding techniques [Bloom97]. However, instead of trying to bracket zero-crossings between sample points where iso-surface functions evaluate to values of opposite sign, we simply find the boundaries in the space between pixel samples of different color. Using the same approaches, we can either point-sample the boundary or compute an approximate mesh representation. In order to increase the precision, we must either use a higher overall resolution or adaptively refine.

One approach is to examine each pair of adjacent cells in 2D or 3D. If the colors are different, the location between the samples is marked as a point on the Voronoi boundary. The operation is very simple and can be accelerated through image operations in graphics hardware.

Another approach is based on a *continuation* method that starts at a point known to be on the boundary and walks along the boundary until all boundary points have been found [Bloom97]. Since we only compare locations near known boundaries, it is output sensitive. The correctness of the continuation method depends on whether the Voronoi boundaries are connected. The boundaries of a generalized Voronoi diagram of a collection of convex sites are always connected, so the method is correct for inputs consisting of point, line-segment, or convex polygonal sites. The method may fail in the presence of curves, curved surfaces, or concave sites where the generalized Voronoi diagram may have isolated components.

In this approach, at least one boundary point must be known as a "seed" value. Assuming convex sites, some Voronoi boundary passes through the edge of the bounded region in which we are computing the diagram, so the method begins by examining every window border pixel. When all Voronoi boundaries are connected only one seed point is needed since all others can be reached from that first point. Starting from a seed point, we recursively check all neighbors that are a different color from the current pixel's. All visited pixels are marked and avoided in the recursion.

This algorithm also finds the *Voronoi neighbors*–pairs of sites that share a Voronoi boundary. This concept is useful in a wide variety of applications, including computing the dual of the ordinary Voronoi diagram–the Delaunay triangulation. The boundary finding algorithms find pairs of adjacent pixels with different colors. The sites corresponding to those two colors are reported to be Voronoi neighbors. Connecting Voronoi neighbors with line segments constructs the Delaunay triangulation.

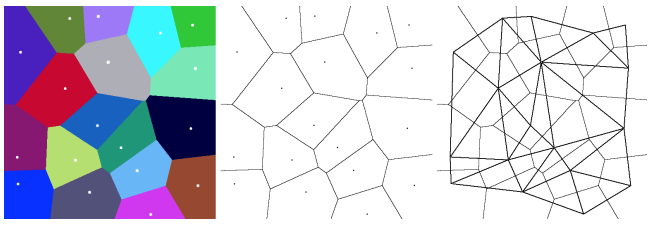


Figure 12: Voronoi diagram of set of 2D points (Left). Boundaries found with continuation-based approach (middle). Delaunay triangulation by connecting neighboring sites (right).

6 SOURCES OF ERROR

In this section we analyze all sources of error in our approach, and discuss how to reduce this error. We consider two broad categories: error in distance approximation and combinatorial error.

6.1 Distance Error

Distance error is the error in the distance computed from a pixel to a site. There are three sources of distance error:

- *Meshing error*, from approximating the true distance function by the distance mesh. We discussed how to bound this error in Section 4.
- *Tessellation error*, from tessellating a curved site into a number of linear sites. The tessellation algorithms presented in [Filip87, Kumar96] give tight bounds. Tessellation error is reduced by using a finer approximation to the site.
- *Hardware precision error*, from the use of fixed-precision arithmetic (integer or floating-point) during rasterization. Hardware precision error cannot be removed without resorting to multiple-precision arithmetic, but hardware error is usually negligible compared to meshing error.

These errors are additive–i.e. the error from one source is not magnified by the other sources. The total distance error is at most the sum of the errors from these three sources.

6.2 Combinatorial Error

Combinatorial error refers to qualitative error as opposed to quantitative. For example, a pixel is assigned the wrong color, or the algorithm reports an incorrect pair of Voronoi neighbors. There are three sources that contribute to combinatorial error:

- *Distance error*, as described in the previous section. With significant distance error, depth comparison at a pixel may make a farther site appear closer, causing the pixel to be colored incorrectly.

- *Resolution error*, a result of discrete sampling. If this sampling is too coarse, we may miss some Voronoi regions or find spurious neighbors. Handling resolution error is described below.
- *Z-buffer precision error*, the limitations of the number of bits of precision provided by the Z-buffer. Current graphics systems have 24 bits or 32 bits of precision for each pixel in the Z-buffer, which is more than the 23 bits provided in standard floating-point. If the distances between two pixels cannot be determined within that precision, the Z-buffer cannot accurately choose the correct color. This effect is small when compared to the other two, but can be significant at very high resolutions with very little distance error. A higher-precision Z-buffer can be simulated in software at a significant loss in efficiency.

Adaptive resolution allows us to "zoom in" on a region of interest, reducing potential resolution error. This involves identifying a window of interest and applying the appropriate linear transformation for zooming into that region. Figure 13 shows an example. Note that when zooming in, sites outside of the viewing region can still have Voronoi regions inside the region. Thus, the "maximum distance to a site" must be adjusted appropriately when computing the distance error bounds.

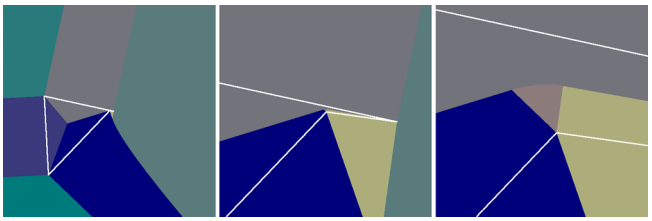


Figure 13: Adaptive resolution allows us to zoom in on features that could otherwise be missed.

Resolution error can cause a number of combinatorial problems, such as missing the entire Voronoi region of a site. One such example is shown in Figure 14 (left two images). When no cell has the color of a particular site, we can separately render the site itself, computing the pixels covering that site. By zooming around those pixels, we will find pixels in the Voronoi region of that site. The same technique can be applied to cells in 3D. Another problem arising from resolution error is incorrectly finding Voronoi neighbors (shown in Figure 14 – right two images). This problem (when due solely to resolution error) can be alleviated by adaptively zooming in on all boundary pixels.

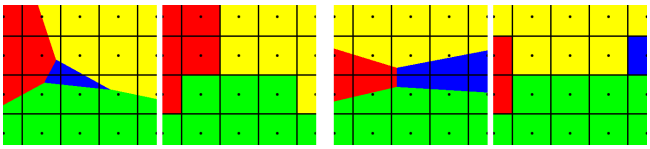


Figure 14: Problems caused by resolution error. An entire region in the center will be missed since it does not hit any pixel centers (left two images). The left and right regions, which should meet, become disconnected after rasterization (right two images).

6.3 Error Bounds

Distance error occasionally causes a pixel to be colored incorrectly. However, in a certain sense, the pixel is "almost" the right color. Assume that there is no Z-buffer precision error, and that we can bound the maximum distance error by ε, as described earlier. For a pixel P colored with the ID of site A and with a computed depth buffer value of D, we know that:

$$D - \varepsilon \le dist(P,A) \le D + \varepsilon$$

Furthermore, we know that for any other site B,

$$D - \varepsilon \le dist(P,B)$$

From this information, we easily determine that

$$dist(P,A) \le dist(P,B) + 2\varepsilon$$

where $dist(X,Y)$ means the distance from the center of pixel X to site Y. That is, if a pixel is colored with the ID of A, then site A is no more than 2ε farther from the pixel center than any other site. The same bound holds in 3D.

7 APPLICATIONS

There are many applications that benefit from fast computation of a discrete Voronoi diagram, an approximation to the distance function, or both. We describe three that we have implemented.

7.1 Motion Planning

Motion planning is a fundamental problem in robotics and computational geometry, with applications to the animation of digital actors, maintainability studies in virtual prototyping, and robot-assisted medical surgery. The classic Piano Mover's problem involves finding a collision-free path for a robot moving from one location (and orientation) to another in an environment filled with obstacles. Numerous approaches to this problem have been proposed, some of which are based on generalized Voronoi diagrams [Latom91]. The underlying idea is to treat the obstacles as sites. The Voronoi boundaries then provide paths of maximal clearance between the obstacles. Due to the practical complexity of computing generalized Voronoi diagrams, the applications of such planners have been limited to environments composed of a few simple obstacles.

Our discrete Voronoi computation algorithm can be applied to motion planning in both static and dynamic environments. The Voronoi algorithm computes the approximate distance to the nearest obstacle. The basic approach we implemented is based on the potential field method, which repels a robot away from the obstacles and towards the goal using a carefully designed artificial potential function. Other Voronoi diagram or distance-based approaches are also possible. The details of our motion planning algorithm are provided in [Hoff99].

We demonstrate our planner's effectiveness in a complex environment: the interior of a house, composed of over 100,000 triangles. We use the x- and y-components of the polygons to give the 2D input primitives for our algorithm. The robot has three degrees of freedom: x- and y-translation along the ground and rotation about the z-axis. Color plate 2 and the video show a sequence of piano motions automatically generated by our motion planner in a static environment. Color plate 2 also shows an image of the distance function for the house. We also apply our planner to environments with moving obstacles. Our video demonstrates the movement of a music stand through a house filled with moving furniture. The entire potential field and the motion planning sequence are computed in real time.

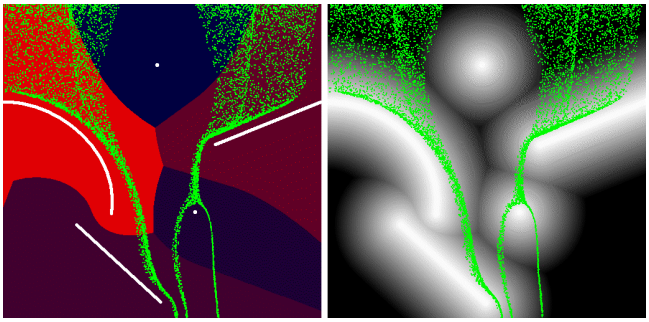


Figure 15: Motion planning of falling particles. Sites are avoided by using the Voronoi diagram's distance buffer (right) to create a potential field. This same principle is used in the rigid-body planner.

7.2 Selection in Complex User Interfaces

Complex 2D user interfaces sometimes require quick determination of the object nearest to the cursor. The Voronoi diagram of the interface can be used as a nearest-object lookup table indexed by sample points. Given the cursor position, it is simple to find the nearest sample point, and thus the nearest object. In some interfaces it may be desirable to know the distance to the selected object as well. We used this technique in our 2D implementation to allow the user to interactively move sites with the mouse.

7.3 Mosaics

We can use our approach for generating Voronoi diagrams to create an interesting artistic effect called mosaicing. A mosaic is a tiled image, where each tile has a single color. The Voronoi diagram of a point set can be used as a tiling [Haebe90]. Each Voronoi tile is colored with a color taken locally from the image. In our implementation, each tile is colored by the image pixel closest to the point site (see color plate 1). Our algorithm can perform this operation very quickly, allowing dynamic mosaics in which the mosaic tiling, the source image, or both may change in real time.

By randomly distributing point sites across an image, we obtain an effect similar to many mosaic filter effects seen in image editing programs. By clustering point sites around areas of higher detail, we obtain a classic tiling seen in many real-life mosaics where smaller tiles are used in areas of greater detail.

8 IMPLEMENTATION

For the 2D case, we implemented a complete interactive system incorporating all of the features and applications described here. Example output is shown throughout the paper. The video demonstrates interactive computation of more complex diagrams. In 3D, we show results from a prototype system that uses a simpler distance meshing strategy (see color plate 3 and the video for example output).

We implemented the 2D and 3D systems in C++ using the OpenGL graphics library and the GLUT toolkit. Any graphics API specification that uses a standard Z-buffered interpolation-based raster graphics system is sufficient to support the Voronoi diagram computation. Motion planning and the basic operations of boundary and neighbor finding require reading back of the color and depth buffers. Our system runs, without source modification, on both an MS-Windows-based PC and a high-end SGI Onyx2 with InfiniteReality Graphics. Surprisingly, the performance on a 400 Mhz Intel Pentium II PC with an Intergraph Intense 3D Pro 3410-T graphics accelerator was comparable to the SGI performance. In fact, in boundary finding, neighbor finding, and particle motion planning applications, the performance exceeded the high-end SGI. This was mainly due to intense buffer readback requirements. Each distance mesh must cover every pixel, so performance is bounded by the graphics hardware's pixel fill-rate. For large numbers of input sites, therefore, the SGI outperforms the PC.

When the distance-error tolerance is relaxed, the amount of geometry rendered for each site can be reduced, slightly improving performance. However, the biggest gains are achieved by reducing the number of pixels filled. In many practical cases, we can increase the performance significantly by bounding the site distance functions to a maximum distance. This allows reduction of the size of the distance meshes drawn so that only a portion of the screen is covered for each site. We exploit this observation to obtain interactive rates in the 1,000-point example shown in color plate 1, in the 10,000-point example shown in the video, and in the general case for the computation of the potential field used in the motion-planner. For closed higher-order primitives, such as polygons, we can further increase performance by restricting the distance function to only the inside or outside regions. This is useful in computing potential fields and medial axes.

9 CONCLUSIONS AND FUTURE WORK

We have presented a method for rapid computation of generalized discrete Voronoi diagrams in two and three dimensions using graphics hardware. We have presented techniques for creating a mesh of the distance function for each site with bounded error, and described how this distance mesh allows us to compute the Voronoi diagram rapidly. We have analyzed various sources of error, as well as how to bound or reduce those errors. Finally, we have demonstrated a few applications using our approach.

In the future, we would like to extend this work in the following ways: generalizations of distance functions and site geometry, further applications, other distance meshing strategies, and more acceleration techniques for the 3D Voronoi volume computation.

ACKNOWLEDGEMENTS

Supported in part by ARO Contract DAAH04-96-1-0257 and DAAG55-98-1-0322, NSF Career Award CCR-9625217, NSF grants EIA-9806027 and DMI-9900157, NIH Research Resource Award 2P41RR02170-13, ONR Young Investigator Award and Intel. We would also like to thank Sarah Hoff for extensive help with editing and color plates, Chris Weigle for suggesting mosaicing using 2D point Voronoi diagrams, the UNC-walkthrough group for the house model, and the reviewers for their helpful comments.

REFERENCES

[Auren91] F. Aurenhammer. *Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure*. ACM Computing Surveys, 23:345–405, 1991.

[Bloom97] J. Bloomenthal, C. Bajaj, J. Blinn, M-P. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc. San Francisco, CA. 1997.

[Chian92] C–S. Chiang. *The Euclidean Distance Transform*. Ph. D. thesis, Dept. Comp. Sci., Purdue Univ., West Lafayette, IN, August 1992. Report CSD-TR 92–050.

[Culve99] T. Culver, J. Keyser, and D. Manocha. *Accurate Computation of the Medial Axis of a Polyhedron*. Proc. of the Fifth Symp. on Solid Modeling and Applications. 1999.

[Dutta93] D. Dutta and C.M. Hoffmann. *On the Skeleton of Simple CSG Objects*. Journal of Mechanical Design, ASME Transactions, 115(1):87–94, 1993.

[Diric50] G.L. Dirichlet. *Uber die Reduktion der Positiven Quadratischen Formen mit Drei Unbestimmten Ganzen Zahlen*. J. Reine Angew. Math., 40:209–27, 1850.

[Filip87] D. Filip and R. Goldman. *Conversion from Bézier-rectangles to Bézier-triangles*. CAD, 19:25–27, 1987.

[Fortu86] S. Fortune. *A Sweepline Algorithm for Voronoi Diagrams*. In Proc. 2nd Annual ACM Symp. on Comp. Geom., pages 313–322, 1986.

[Goldf89] J. Goldfeather, S. Molnar, G. Turk, and H. Fuchs. *Near Real-time CSG Rendering Using Tree Normalization and Geometric Pruning*. IEEE Computer Graphics and Applications, 9(3):20–28, May 1989.

[Haebe90] P. Haeberli. *Paint by Numbers: Abstract Image Representation*. Computer Graphics (SIGGRAPH '90 Proc). vol. 25. pgs 207-214.

[Held97] M. Held. *Voronoi Diagrams and Offset Curves of Curvilinear Polygons*. Computer-Aided Design, 1997.

[Hoff99] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. *Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware*. Technical Report TR99-011, Dept. of Comp. Sci., University of North Carolina at Chapel Hill, 1999.

[Hoffm94] C.M. Hoffmann. *How to Construct the Skeleton of CSG Objects*. In A. Bowyer and J. Davenport, editors. Proc. of the Fourth IMA Conference, The Mathematics of Surfaces, University of Bath, UK, Sept. 1990. Oxford University Press, New York, 1994.

[Inaga92] H. Inagaki, K. Sugihara, and N. Sugie. *Numerically Robust Incremental Algorithm for Constructing Three-dimensional Voronoi Diagrams*. In Proc. 4th Canad. Conf. Comp. Geom., pgs 334-339, 1992.

[Kumar96] S. Kumar, D. Manocha, and A. Lastra. *Interactive Display of Large NURBS Models*. IEEE Trans. on Vis. and Computer Graphics. vol 2, no 4, pgs 323-336, Dec 1996.

[Latom91] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[Laven92] D. Lavender, A. Bowyer, J. Davenport, A. Wallis, and J. Woodwark. *Voronoi Diagrams of Set-theoretic Solid Models*. IEEE Computer Graphics and Applications, 12(5):69–77, Sept 1992.

[Lee82] D.T. Lee. *Medial Axis Transformation of a Planar Shape*. IEEE Trans. Pattern Anal. Mach. Intell., PAMI–4:363–369, 1982.

[Lengy90] J. Lengyel, M. Reichert, B.R. Donald, and D.P. Greenberg. *Real-time Robot Motion Planning Using Rasterizing Computer Graphics Hardware*. Computer Graphics (SIGGRAPH '90 Proc.), vol. 24, pgs 327–335, Aug 1990.

[Milen93] V. Milenkovic. *Robust Construction of the Voronoi Diagram of a Polyhedron*. In Proc. 5th Canadian. Conference on Comp. Geom., pgs 473-478, 1993.

[Milen93b] V. Milenkovic. *Robust Polygon Modeling*. Computer Aided Design, 25(9), 1993. (special issue on Uncertainties in Geometric Design).

[Okabe92] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.

[Rossi92] J. Rossignac, A. Megahed, and B. Schneider. *Interactive Inspection of Solids: Cross-sections and Interferences*. Computer Graphics (SIGGRAPH '92 Proc.), vol. 26, pgs 353–360, July 1992.

[Rossi86] J.R. Rossignac and A.A.G. Requicha. *Depth-buffering Display Techniques for Constructive Solid Geometry*. IEEE Computer Graphics and Applications, 6(9):29–39, 1986.

[Sheeh95] D.J. Sheehy, C.G. Armstrong, and D.J. Robinson. *Computing the Medial Surface of a Solid from a Domain Delaunay Triangulation*. In Proc. ACM/IEEE Symp. on Solid Modeling and Applications, May 1995.

[Shamo75] M.I. Shamos and D.Hoey. *Closest-point Problems*. In Proc. 16th Annual IEEE Symposium on Foundations of Comp. Sci., pages 151–162, 1975.

[Sugih94] K. Sugihara and M. Iri. *A Robust Topology-oriented Incremental Algorithm for Voronoi Diagrams*. International Journal of Comp. Geom. Appl., 4:179–228, 1994.

[Sherb95] E.C. Sherbrooke, N.M. Patrikalakis, and E. Brisson. *Computation of the Medial Axis Transform of 3D Polyhedra*. In Solid Modeling, pages 187–199. ACM, 1995.

[Teich97] M. Teichmann and S. Teller. *Polygonal Approximation of Voronoi Diagrams of a Set of Triangles in Three Dimensions*. Tech Rep 766, Lab of Comp. Sci., MIT, 1997.

[Vleug95] J. Vleugels and M. Overmars. *Approximating Generalized Voronoi Diagrams in Any Dimension*. Technical Report UU-CS-1995-14, Dept. of Comp. Sci., Utrecht University, 1995.

[Vleug96] J. Vleugels, V. Ferrucci, M. Overmars, and A. Rao. *Hunting Voronoi Vertices*. Comp. Geom. Theory Appl., 6:329–354, 1996.

[Voron08] G.M. Voronoi. *Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. Deuxième Mémoire: Recherches sur les Parallélloèdres Primitifs*. J. Reine Angew. Math., 134:198–287, 1908.

[Woo97] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*, Second Edition. Addison Wesley, 1997.

Plate 1: We can use the Voronoi diagram to create real-time mosaic effects. Left: Voronoi diagram computed for 1000 2D point sites at a resolution of 512x512. Center: Same diagram using the point locations to index into a 256x256 face texture to obtain the region colors. Right: The final effect at 512x512 with points removed. In the video, we show this same effect in real-time with 10,000 points.

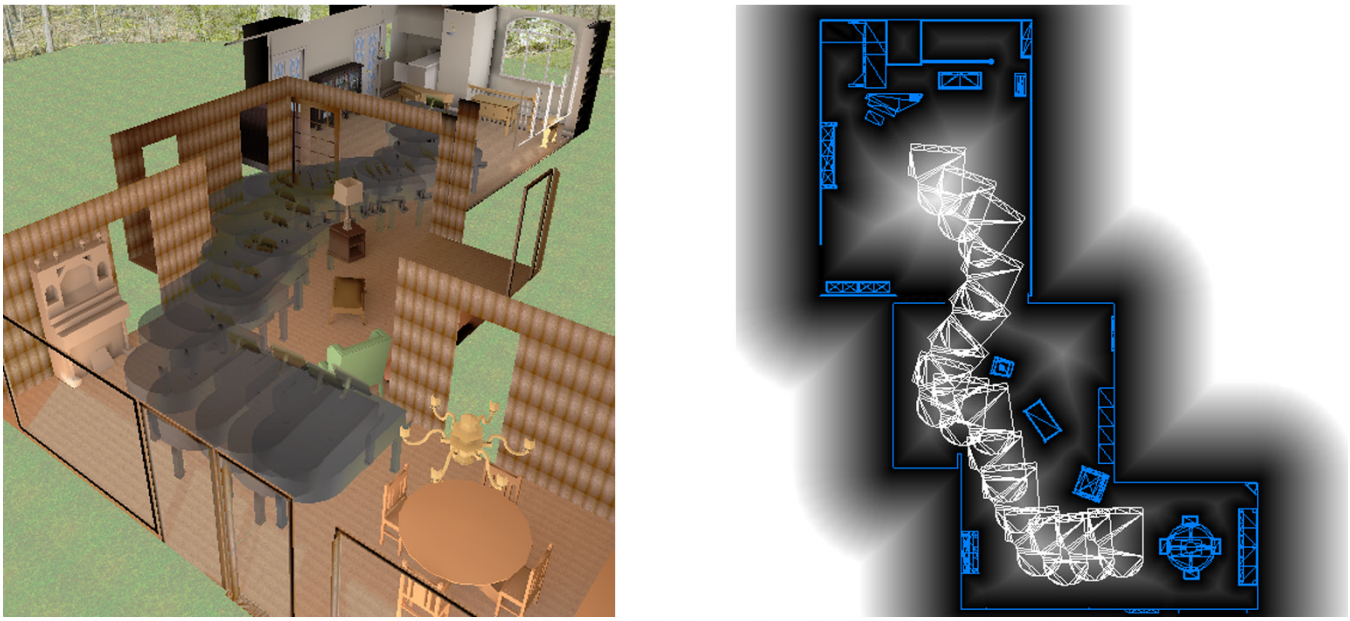


Plate 2: We create a potential field from the Voronoi diagram of the house floorplan to plan the motion of a piano through a complex static scene in real-time. Left: The piano motion sequence. Right: Motion sequence overlaying the floorplan distance function. The video demonstrates navigation through a dynamic scene in real-time.

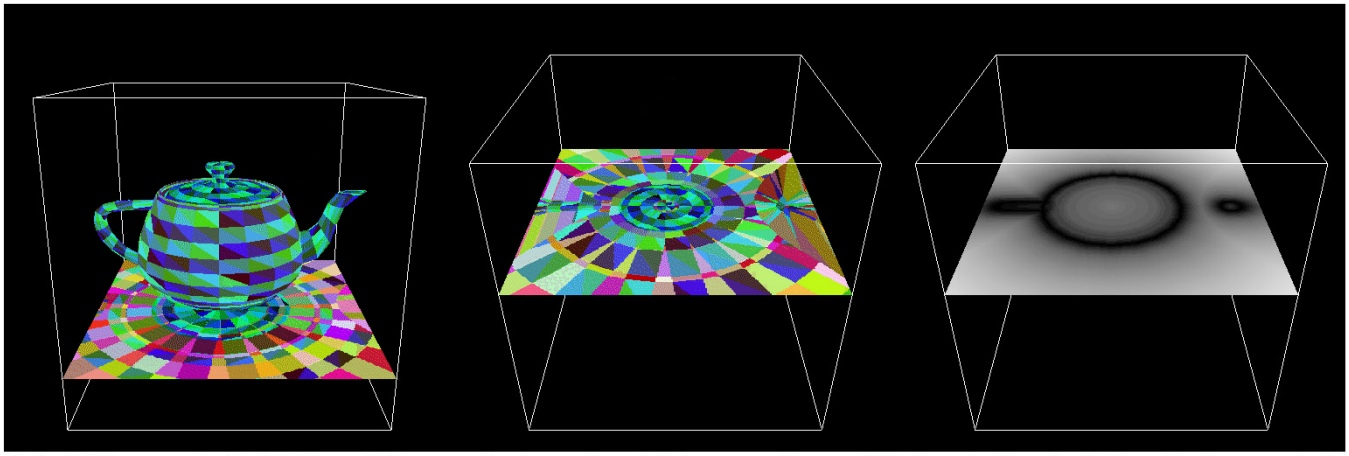


Plate 3: We compute the 3D Voronoi diagram of polygonal models by computing one 2D slice at a time. Left: One slice is computed just below the teapot. Center: Complete slice halfway through. Right: Distance image. The colors indicate the Voronoi regions for face edge, and vertex sites. Only the face site colors are shown on the model.