Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware

paper by Kennet E. Hoff et al.
(University of North Carolina at Chapel Hill)

presented by Daniel Emmenegger
GDV-Seminar ETH Zürich, SS2000
What is a Voronoi Diagram?

Given a collection of geometric primitives, it is a subdivision of space into cells (regions) such that all points in a cell are closer to one primitive than to any other.
Ordinary vs. Generalized

<table>
<thead>
<tr>
<th>Ordinary</th>
<th>Generalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitives: Points</td>
<td>Primitives: Points, Lines, Polygons, Curves, ...</td>
</tr>
<tr>
<td>Nearest Euclidean Distance</td>
<td>Varying distance metrics</td>
</tr>
</tbody>
</table>

Higher-order Sites
Voronoi Boundary

Curves forming the boundary between the various cells

frontier between two cells of different color
Delaunay-Triangulation

Duality structure to Ordinary Voronoi Diagrams:

Connect all primitives with their nearest neighbors
### What can we do with this stuff?

**FUNDAMENTAL CONCEPT:**

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1644</td>
<td>Descartes</td>
<td>Astronomy</td>
</tr>
<tr>
<td>1850</td>
<td>Dirichlet</td>
<td>Math</td>
</tr>
<tr>
<td>1908</td>
<td>Voronoi</td>
<td>Math</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1970</td>
<td>divers...</td>
<td>Computational geometry and related areas</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
What can we do with this stuff?

Nearest Site  Maximally Clear Path

Density Estimation  Nearest Neighbors
Motivation

- **Previous Work: Exact Algorithms**
  no error but ...
  - Boundaries composed of high-degree curves and surfaces and their intersections
  - Complex and difficult to implement
  - Robustness and accuracy problems

- **Previous Work: Approximate Algorithms**
  provide a practical solution but...
  - Difficult to error-bound
  - Restricted to static geometry
  - Relatively slow
Approximate generalized Voronoi Diagram computation with the following features:

- Easily generalized
- Efficient and practical
- Has tight bounds of accuracy
- Simple to understand and implement
Formal Definition

Set of input sites (primitives) $A_1, A_2, \ldots, A_k$

$\text{dist}(p, A_i)$: distance from the point $p$ to the site $A_i$

The dominance region of $A_i$ over $A_j$ is defined by

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

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

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

Partition of space into $V(A_1), V(A_2), \ldots, V(A_k)$:

**Generalized Voronoi Diagrams**
Discrete Voronoi Diagrams

Uniformly point-sample the space containing Voronoi sites
For each sample find closest site and its distance

Brute-force-Algorithm:

- iterate through all samplepoints (cells)
- iterate through all primitives => HARDWARE
Basic Idea: Cones

To visualize Voronoi Diagrams for points ...

topview, parallel  perspective view
Graphics Hardware Acceleration

Simply rasterize the cones using graphics hardware

Our 2-part discrete Voronoi diagram representation

Origin: Woo97
## Basic Idea: Distance Function

Render a polygonal mesh approximation to each site’s distance function.

Each site has:
- unique color ID assigned
- corresponding distance mesh rendered in this color using parallel projection

We make use of:
- linear interpolation across polygons
- Z-Buffer depth comparison operation
The Distance Function

And Polygons, Bezier-Curves, ... What is their distance function?

Compose them of points and lines!
Approximation Error

Distance Function: Meshing Error of a cone

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

Evaluate distance at each pixel for all sites
Accelerate using graphics hardware
Approximation of the Distance Function

- Avoid per-pixel distance evaluation
- Point-sample the distance function
- Reconstruct by rendering polygonal mesh

Point
Line
Triangle
Sweep apex of cone along higher-order site to obtain the shape of the distance function
Tessellate curve into a polyline
Tessellation error is added to meshing error
**Algorithm A**: (very simple, accelerated through image op. in ghw)
- examine each pairs of adjacent cells
- if color different, location between is marked as boundary-point

**Algorithm B**: continuation method
- choose seed (known point of boundary)
- walk along boundaries until all boundary points are found
Main Question:
Which colors touch in the image?

Answer, how to find them:
Same algorithm as used for finding boundaries
What about 3D?

Slices of the distance function for a 3D point site

Distance meshes used to approximate slices
What about 3D?

Graphics hardware can generate one 2D slice at a time

Point sites
### 3D Distance Functions

<table>
<thead>
<tr>
<th>Point</th>
<th>Line segment</th>
<th>Triangle</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Point" /></td>
<td><img src="image" alt="Line segment" /></td>
<td><img src="image" alt="Triangle" /></td>
</tr>
<tr>
<td><img src="image" alt="1 sheet of a hyperboloid" /></td>
<td><img src="image" alt="Elliptical cone" /></td>
<td><img src="image" alt="Plane" /></td>
</tr>
</tbody>
</table>

- **Point**: A single point in 3D space.
- **Line segment**: A line segment connecting two points.
- **Triangle**: A triangular face.
- **1 sheet of a hyperboloid**: A hyperbolic surface.
- **Elliptical cone**: A cone with an elliptical base.
- **Plane**: A flat surface that extends infinitely in all directions.
Sources of Error

• **Distance Error**
  – meshing
  – tessellation
  – hardware precision

• **Combinatorial Error**
  – Z-Buffer precision
  – distance
  – pixel resolution
Resolution Error

Adaptive Resolution

zoom in to reduce resolution error
Error bound is determined by the pixel resolution 

\[ \varepsilon \leq \text{farthest distance a point can be from a pixel sample point} \]
Assume: no Z-Buffer precision error

we can bound the maximum distance error by $\varepsilon$

for a pixel $P$ colored with ID of site (primitive) $A$ and with computed depth buffer of value $D$, we know:

$$D - \varepsilon \leq \text{dist}(P, A) \leq D + \varepsilon$$

further we know, for any other site $B$

$$D - \varepsilon \leq \text{dist}(P, B)$$

With this information we easily determine that

$$\text{dist}(P, A) \leq \text{dist}(P, B) + 2\varepsilon$$
Implementation

- complete interactive system in 2D
  - written in C++ using OpenGL and GLUT
  - a standard Z-buffered interpolation-based raster graphics system
- some first prototypes in 3D
- runs (without source modification) on:
  - MS-Windows-based PC
  - high-end SGI Onxy2
- several problem-based modifications to increase performance...
Demo

VIDEO
Conclusions

• General:
  – Idea is originally not from E. Hoff or one of the other writers => Open GL Programming Guide, 2nd Edition M. Woo et al.

• My opinion:
  – Concept very easy to understand...
  – ...but the main idea is not immediately obvious!
  – All ideas are implemented, so the reader can easily determine if everything (the notion of distance function etc.) really works
Questions?

http://www.cs.unc.edu/~hoff/