Geometric Modeling in Graphics

Part 8: Volumes

Martin Samuelčík
www.sccg.sk/~samuelcik
samuelcik@sccg.sk
Volumes

- 3D set of points, embedded in space $E^3$
- Representing also interior of object
- Discrete grid
  - Sampling function values in grid points
  - Function – binary, distance, intensities, function values, distance vectors, axial distance vectors, physical properties, …
  - Grid – uniform, octree, tetrahedral, …
- Parametric volumes
  - Set of all points $X \in E^3$ such that $X = f(u,v,w)$, $u \in <u_0,u_1>$, $v \in <v_0,v_1>$, $w \in <w_0,w_1>$
- FREP
  - Set of all points $X \in E^3$ such that $f(X) \leq 0$
Volumes

Geometric Modeling in Graphics
Distance function

- Function defined as distance of point to object
  - $d: \mathbb{R}^3 \rightarrow \mathbb{R}^+$
- For set $\Sigma$, distance function without sign is
  \[
  \text{dist}_\Sigma(p) = \inf_{x \in \Sigma} \|x - p\|.
  \]
- Extension - distance vectors
- Distance to object with sign
  \[
  d_S(p) = \text{sgn}(p) \inf_{x \in \partial S} \|x - p\|,
  \]
  where
  \[
  \text{sgn}(p) = \begin{cases} 
  -1 & \text{if } p \in S \\
  1 & \text{otherwise}.
  \end{cases}
  \]
Distance function
Distance function

- Isosurface for isovalue $\tau$: $\{p | d(p) = \tau\}$
- Surface, boundary of object is isosurface for isovalue 0
- Vector of first order derivative, gradient $\|\nabla d\| = 1$
  - Perpendicular to isosurface at point – normal approximation
- Hessian

$$H = \begin{pmatrix}
    d_{xx} & d_{xy} & d_{xz} \\
    d_{yx} & d_{yy} & d_{yz} \\
    d_{zx} & d_{zy} & d_{zz}
\end{pmatrix}$$

- Mean curvature

$$\kappa_M = \frac{1}{2} (d_{xx} + d_{yy} + d_{zz})$$

- Gauss curvature

$$\kappa_G = \begin{vmatrix}
    d_{xx} & d_{xy} \\
    d_{yx} & d_{yy}
\end{vmatrix} + \begin{vmatrix}
    d_{xx} & d_{xz} \\
    d_{zx} & d_{zz}
\end{vmatrix} + \begin{vmatrix}
    d_{yy} & d_{yz} \\
    d_{zy} & d_{zz}
\end{vmatrix}$$
Distance function

- Distance function is continuous – $C^0$
- Problem points – points that have same distance from at least two different points on object’s surface – **cut locus**
- Function is $C^1$ except points from cut locus
- For $C^k$ surface, distance function is $C^k$ in some neighborhood of point on surface
### Discretization

- Distance function sampling – distance field
- Topology of sample points
  - Uniform grid
  - Octree
  - Tetrahedral grid
  - Octahedral grid
- Voxel, Cell – volume element of sampling grid
- Voxelization
- Criterion of representation – distance of all cut locus points is larger than sampling resolution
- Approximating gradient

\[
\begin{align*}
g_{i,j,k}^x &= d_{i+1,j,k} - d_{i-1,j,k} \\
g_{i,j,k}^y &= d_{i,j+1,k} - d_{i,j-1,k} \\
g_{i,j,k}^z &= d_{i,j,k+1} - d_{i,j,k-1} \\
n_{i,j,k} &= \frac{g_{i,j,k}}{\|g_{i,j,k}\|}.
\end{align*}
\]

**Geometric Modeling in Graphics**
Voxelization

- Given surface of object S
- Definition and placement of grid points
- Topology of grid points – shape of grid
- For each grid point G, computation of distance G from S
- Based on representation of S
  - Polyhedral mesh
    - Point-polygon distance computation
  - Implicit surface
    - Direct approximation, numerical solution
  - Parametric surface
    - Numerical methods for distance computation
- Computation only near surface of S, then fast propagation of distance values to remaining grid points
Mesh voxelization

- Given closed triangular 2-manifold mesh
- Choosing triangle that is closest to given grid point
  - Optimization using bounding volumes, grid, octrees
- Computation of grid point – triangle distance
- 7 cases of grid point projection to triangle plane

- given triangle ABC, given grid point S, looking for distance DST
- using barycentric coordinates of point X in triangle plane, \( X = uA+vB+wC, u+v+w=1 \)
- let \( a=(A-C,A-C), b=(A-C,B-C), c = (B-C,B-C), d = (A-C,C-S), e=(B-C,C-S), f=(C-S,C-S) \)
- let \( SX \) is projection of S to plane of triangle ABC, \( SX=sA+tB+(1-s-t)C \)
- \( s=(be-cd)/(ac-b^2), t=(bd-ae)/(ac-b^2) \)
- if \( 0\leq s\leq 1, 0\leq t\leq 1, 0\leq 1-s-t\leq 1 \), then \( SX \) is inside triangle ABC, and \( VZD=|S,SX| \)
- else if \( s<0 \) or \( s>1 \), then find point \( SXS \) as projection of point \( SX \) to line BC
  - \( SXS=pB+(1-p)C, p=(SX-C,B-C)/(B-C,B-C) \)
  - if \( p<0 \), then \( VZD=|S,C| \)
  - if \( 0\leq p\leq 1 \), then \( VZD=|S, SXS| \)
  - if \( p>1 \), then \( VZD=|S,B| \)
- similarly for \( t<0, t>1, (1-s-t)<0, (1-s-t)>1 \)
Mesh voxelization

- Local methods
- Computing exact distance only for grid points in close vicinity of mesh surface
- Extruded objects for vertices, edges and triangles of mesh
- Identifying grid points lying inside extruded objects
- Simple computation of distance for points inside extruded objects
Mesh voxelization

- Sign computation
- Determining if grid point is inside or outside of object
- Number of intersections between arbitrary ray from grid point and mesh boundary
  - Odd number of intersections – inside
  - Even number of intersection – outside
- For $C^1$ surfaces, dot product of normal and distance vector
  - If dot product is positive, grid point is inside
  - Mesh – not $C^1$ – using angle-weighted pseudo-normals for edges and vertices of mesh
Distance transforms

- Propagation of distance values in computed grid points to remaining unprocessed grid points

Grid propagation:
- Sweeping – uniform slices propagation
- Wavefront – from surface to higher distances

Computation for voxel:
- Chamfer:
  - New distance in grid point is computed from already known distances in neighboring grid points
- Vector:
  - New distance vector in grid point is computed from already known distance vectors in neighboring grid points
- Eikonal:
  - Distance in grid points is filled using iterative solution of differential equation
Distance transforms

- Initialization – computation of distance for grid points near surface of object
- Chamfer methods
  - Sweeping

```c
/* Forward Pass */
FOR(z = 0; z < f_z; z++)
  FOR(y = 0; y < f_y; y++)
    FOR(x = 0; x < f_x; x++)
      \[ F[x,y,z] = \inf_{i,j,k \in f_p} (F[x+i,y+j,z+k]+m[i,j,k]) \]

/* Backward Pass */
FOR(z = f_z-1; z >= 0; z--)
  FOR(y = f_y-1; y >= 0; y--)
    FOR(x = f_x-1; x >= 0; x--)
      \[ F[x,y,z] = \inf_{i,j,k \in b_p} (F[x+i,y+j,z+k]+m[i,j,k]) \]
```

<table>
<thead>
<tr>
<th>Transform</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>City Block (Manhattan)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chessboard</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-Euclidean 3 × 3 × 3</td>
<td>1</td>
<td>(\sqrt{2})</td>
<td>(\sqrt{3})</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete Euclidean 3 × 3 × 3</td>
<td>1</td>
<td>(\sqrt{2})</td>
<td>(\sqrt{3})</td>
<td>(\sqrt{5})</td>
<td>(\sqrt{6})</td>
<td>3</td>
</tr>
<tr>
<td>&lt;a, b, c &gt;_{opt} 3 × 3 × 3</td>
<td>0.92644</td>
<td>1.34065</td>
<td>1.65849</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-Euclidean 5 × 5 × 5</td>
<td>1</td>
<td>(\sqrt{2})</td>
<td>(\sqrt{3})</td>
<td>(\sqrt{5})</td>
<td>(\sqrt{6})</td>
<td>3</td>
</tr>
</tbody>
</table>
```

- Wavefront – priority queue for grid points with minimal distance
Fast marching method

- Eikonal distance transform
- Simulating expanding surface with constant speed – inflating balloon
- Time of surface (balloon) arrival to grid point – distance
- $T$ – time of arrival to grid point $x$
- $F$ – speed of surface expansion in $x$
  - $F$ is constant

\[ ||\nabla T(x)|| F(x) = 1 \]
Fast marching method

- „frozen” point – final distance was computed for point
- „narrow band“ point – there is some distance computed, but is not final
- H – set of „narrow band“ points, priority queue

Initialization()
{
    for each voxel v in I
    {
        Freeze v;
        for each neighbour vn of v
        {
            compute distance d at vn;
            if vn is not in narrow band
            {
                tag vn as narrow band;
                insert (d,vn) in H;
            }
            else
                decrease key of vn in H to d;
        }
    }
}

Loop()
{
    while H is not empty
    {
        Extract v from top of H;
        Freeze v;
        for each neighbour vn of v
        {
            if vn is not frozen
            {
                compute distance d at vn;
                if vn is not in narrow band
                {
                    tag vn as narrow band;
                    insert (d,vn) in H;
                }
                else
                    decrease key of vn in H to d;
            }
        }
    }
}
Fast marching method

Computation of distance for grid point from neighboring point distances using constant gradient

\[
\frac{1}{F^2} = \begin{cases} 
\max(D_{-2x}^{-x} G, -D_{2x}^{+x} G, 0)^2 + \\
\max(D_{-2y}^{-y} G, -D_{2y}^{+y} G, 0)^2 + \\
\max(D_{-2z}^{-z} G, -D_{2z}^{+z} G, 0)^2 
\end{cases}
\]

\[
||\nabla T||^2 = \begin{cases} 
\max(V_A - V_B, V_A - V_C, 0)^2 + \\
\max(V_A - V_D, V_A - V_E, 0)^2 + \\
\max(V_A - V_F, V_A - V_G, 0)^2 
\end{cases}
\]
Parametric surface voxelization

- Conversion to polyhedral or implicit representation
- Minimization of \( d(u, v) = \| S(u, v) - p \| \) using numerical iterative solutions
Implicit surface voxelization

- Isosurface of function \( \{ X \in E^3; f(X) = 0 \} \)
- For some surface, it is sufficient to sample just \( f \)
- Sampling function \( \frac{f}{||\nabla f||} \)
- Iteratively find closest point to grid point on implicit surface in the gradient direction
  - Let \( (x_0, y_0, z_0) \) is given grid point
  - \( (x_{i+1}, y_{i+1}, z_{i+1}) = (x_i, y_i, z_i) - \frac{f(x_i, y_i, z_i)}{f_x(x_i, y_i, z_i)^2 + f_y(x_i, y_i, z_i)^2 + f_z(x_i, y_i, z_i)^2} (f_x(x_i, y_i, z_i), f_y(x_i, y_i, z_i), f_z(x_i, y_i, z_i)) \)
  - Finish when one iteration does not change position of approximation so much
Interpolation

- Approximation of distance function for arbitrary space point from grid values - interpolating grid values

Nearest neighbor interpolation
- Given space point $C$, find grid point $G$ that is closest to $C$
- $d(C) = d(G)$

Trilinear interpolation
- Given space point $C$, find voxel $V$ where it is located
- Compute $C$ as linear combination $V$'s corner points
- $\lambda_1 = (C - C_{000}, C_{100} - C_{000})$
- $\lambda_2 = (C - C_{000}, C_{010} - C_{000})$
- $\lambda_3 = (C - C_{000}, C_{001} - C_{000})$
- $C = w_{000}C_{000} + w_{100}C_{100} + \ldots + w_{111}C_{111}$
- $w_{ijk} = (1 - \lambda_1)^{1-i}\lambda_1^i (1 - \lambda_2)^{1-j}\lambda_2^j (1 - \lambda_3)^{1-k}\lambda_3^k$
- $w_{000} + w_{100} + \ldots + w_{111} = 1$
- $d(C) = w_{000}d(C_{000}) + w_{100}d(C_{100}) + \ldots + w_{111}d(C_{111})$

Tricubic interpolation
Visualization

- Conversion to other representations
  - Polyhedral representation – marching cubes
  - Point clouds – projections of grid point onto surface in the opposite direction of gradient

- Direct visualization
  - Slicing

- Raytracing
  - Traversal of grid along ray
  - Finding first voxel containing isosurface
  - Using distance function interpolation to find more accurate intersection
  - Using subsampled values for finer approximation

- Points sampling
  - Approximation of closest point on surface \[ p_f = p - \nabla d_S(p) d_S(p) \]
Raytracing

Marching cubes

- Generating set of triangles that approximate isosurface of distance function for given isovalue $\tau$
- Generating triangles for each voxel separately
  - Get 8 grid values in corners of voxel
  - Mark each corner $C$ as inside or outside by comparing distance value in corner and isovalue $\tau$
    - Outside - $d(C) \geq \tau$
    - Inside - $d(C) < \tau$
  - For each edge $AB$ of voxel, if it connect inside and outside corner, construct edge vertex using linear interpolation
    $$V_{AB} = \frac{d(B) - \tau}{d(B) - d(A)} A + \frac{\tau - d(A)}{d(B) - d(A)} B$$
    - Interpolating gradients in $A, B$ to get normal in $V_{AB}$
  - Connect all edge vertices in voxel forming several triangles based on configuration of inside and outside corners
Marching cubes

- Basic configurations of inside, outside corners

- 256 total configurations (rotation, mirroring)

- Implementation
  - http://paulbourke.net/geometry/polygonise/
  - Preparing vertex code – marking each corner as inside or outside, 8bit
  - Computing edge vertex for each of 12 edges, if necessary
  - Connecting edge vertices into triangles based on vertex code, using lookup table with 256 records, each record has list of edge indices pointing to edge vertices
**Skeleton, medial axis**

- Simple primitives (line segments) representing shape of whole object with same topological properties
- Detecting cut locus points – discontinuities in derivation of distance function
- Finding extremal values inside object
- Comparison of distance vectors
- Used for skeleton animation, parametrization, …
Fonts representation

- Using 2D distance field for representation of each glyph
- More precise representation of border
- Easier rendering of border effects

(a) 64x64 texture, alpha-blended
(b) 64x64 texture, alpha tested
(c) 64x64 texture using our technique
Fonts representation

(a) High resolution input

(b) 64x64 Distance field

Geometric Modeling in Graphics
Morphing

- Interpolation between two objects in time
- Compacting representation of given objects – same sampling grid points for both representations
- Linear interpolation of two values in each grid points
Morphology

- Operations for discrete signal processing
- Erosion \( X \ominus B = \{p | B_p \subseteq X\} \)
- Dilatation \( X \oplus B = \{p | B_p \cap X \neq \emptyset\} \)
- Opening \( X \bullet B = (X \oplus B) \ominus B \)
- Closing \( X \circ B = (X \ominus B) \oplus B \)
CSG operations

- Simple and fast Boolean operations on two objects
- Distance fields of objects must be compacted – must have same sampling grid points
- Approximation and alias near sharp features
- **Union** - $D = \min(D_1, D_2)$
- **Intersection** - $D = \max(D_1, D_2)$
- **Difference** - $D = \max(D_1, -D_2)$
CSG operations repair

- [link](http://www.sccg.sk/~novotny/doc/vg05.pdf)
- Improvement of representation after Boolean operation
- Detecting and repairing distance function near sharp features with insufficient sampling density
Hypertextures

- Adding rendering details over object surface
- Defining region over surface for texture mapping

\[ D(p) = \begin{cases} 
1 & \text{if } d(p)^2 \leq r_i^2 \\
0 & \text{if } d(p)^2 \geq r_o^2 \\
\frac{r_o^2-d(p)^2}{r_o^2-r_i^2} & \text{otherwise},
\end{cases} \]

- Using \( D(p) \) to obtain data from 3D texture or to generate other properties such like direction, density, tangent plane
Object modeling

- Surface smoothing
- Subdivision
- Parametrization
- Error computation
- Objects comparison
- Collision detection
- Simulations, animation
- Reconstruction

After 92s

After 1932s

After 3680s

After 17020s
The End for today

Geometric Modeling in Graphics