Implicit Surfaces

An implicit surface is simply an iso-contour of a scalar function $f(x,y,z)=0$.

The term is usually used when modeling a surface, whereas iso-contour is used when visualizing a scalar field.

Same thing!!!

Point-based Modeling primitives
- Blobbies
- Meta-balls
- Soft Objects

Jim Blinn, 1982

$f(x, y, z) = \sum_k b_k e^{-a_k (x^2 + y^2 + z^2)} - T$

where, if $b_k > 0$ => bump
if $b_k < 0$ => dent

Smooth function - Gaussian
Every blobby affects the shape everywhere.
- No finite support
van der Waals surface

Meta-balls

Examples

Soft Objects

Molecular Models

van der Waals, et al., piecewise quadratic spline
Implicit Surfaces

How to render?
- Have an analytical function.
- Sample it on a regular grid
- Use marching cubes to extract a polygonal surface.
- Can also use ray-casting/marching to sample the space.
  - Root-finding problem to determine the surface.

Rendering

- So, implicit surfaces are a way of defining a 3D volume.
- Can use splatting or any other volume rendering technique to display them by resampling to a regular volume grid.

Convolution Surfaces

- Bloomenthal proposed using curves and 2D surfaces as primitives, rather than simple points.
  - Smooths the transition between points.
  - He proposed a convolution of the basis function with a continuous curve.
Convolution Surfaces

- Discretely

Convolution Surfaces

- Poorly spaced meta-balls

Convolution Surfaces

- Logically
Convolution Surfaces

Rule of thumb
- If spacing is less than radius or basis normalization, as spacing is decreased, the weights need to be adjusted to preserve the thickness. Splatting or reconstruction has a fixed spacing and uses a kernel appropriate for this spacing.

2D Contouring
- Continuous \( f(x,y) \)
  - Use steepest decent to find zero crossing (root) of the function \( f(x,y) - c \)
  - Follow contour from this seed point until we reach a boundary or loop back.
  - Direction close to \( \nabla f \otimes z \)

Examples
- Hand model generated from convolution of Winslow data with \( f(x,y) \)
2D Contouring

- Given a quadrilateral
  - \( f(x,y) = 0.5 \)

Slide 22

2D Contouring

- Discrete Data
  - Assume the Mean Value Theorem
  - Assume monotonicity?
  - 1D Analogy
    - 5 Points

Slide 23

3D Contouring

- Treat volume as a set of 2D slices
  - Apply 2D Contouring algorithm on each slice.
  - Or given as a set of hand-drawn contours
  - Stitch the slices together.

Marching Cube - The Problem

- Extracting an iso-surface from an implicit function, that is, extracting a surface from volume data (discrete implicit function) \( f(x,y,z) = T \).
Contour Stitching

Problem:
Given: 2 two-dimensional \textit{closed} curves
Curve #1 has \( m \) points
Curve #2 has \( n \) points
Which point(s) does vertex \( i \) on curve one correspond to on curve two?

Marching Cubes

- Lorensen and Cline, SIGGRAPH ‘87
- Predominant method used today.
- Efficient and simple

Marching Cubes

- Treat each cube individually
  - No 2D contour curves
- Allow intersections only on the edges or at vertices.
- Pre-calculate all of the necessary information to construct a surface.

Marching Cubes

- Consider a single cube
  - All vertices above the contour threshold
  - All vertices below
  - Mixed above and below
Marching Cubes

- Binary label each node => (above/below)
- Examine all possible cases of above or below for each vertex.
- 8 vertices implies 256 possible cases.

Marching Cubes

- 14 unique cases
  - +/- symmetry
  - rotational symmetry
  - mirror symmetry

Marching Cube - Summary

- Create a cube
- Classify each voxel
- Build an index
- Lookup edge list
- Interpolate triangle vertices
- Calculate and interpolate normals

Step 1: Create a Cube

- Consider a cube defined by eight data values: four from slice K, and four from slice K+1
**Step 2: Classify Each Voxel**

- Binary classification of each vertex of the cube as to whether it lies
  - outside the surface (voxel value < isosurface value)
  - or inside the surface (voxel value <= isosurface value).

**Step 3: Build an Index**

- Use the binary labeling of each voxel to create an 8-bit index. (8 vertex - 256 cases)

**Step 5: Interpolate Triangle Vertices**

- For each edge, find the vertex location along the edge using linear interpolation of the voxel values.

**Step 6: Compute Normals**

- Calculate the normal at each cube vertex
- Use linear interpolation to interpolate the polygon vertex normal
Ambiguities

- Right or wrong?

Marching Cubes

- Topological inconsistencies in the 15 cases
  - Turns out positive and negative are not symmetric.

A Bad Example

- Animating the contour value
- Special functions for contouring
- Varying speeds and numbers of triangles
Marching Cubes

- Data Structures/Tables

```c
static int const HexaEdges[12][2] = {{0,1}, {1,2}, {2,3}, {3,0}, {4,5}, {5,6}, {6,7}, {7,4}, {0,4}, {1,5}, {3,7}, {2,6}};

typedef struct {
  EDGE_LIST HexaEdges[16];
} HEXA_TRIANGLE_CASES;

/* Edges to intersect. Three at a time form a triangle. */
static const HEXA_TRIANGLE_CASES HexaTriCases[] = {
  { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }, /* 0 */
  {  0,  8,  3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }, /* 1 */
  {  0,  1,  9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }, /* 2 */
  {  1,  8,  3,  9,  8,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }, /* 3 */
  ...
```

Marching Cubes - How simple

```c
/* Determine the marching cubes index */
for ( i=0, index = 0; i < 8; i++)
  if (val1[nodes[i]] >= thresh) /* If the nodal value is above the */
    index |= CASE_MASK[i]; /* threshold, set the appropriate bit. */

triCase = HexaTriCases + index; /* triCase indexes into the MC table. */
edge = triCase->HexaEdges; /* edge points to the list of intersected edges */

for ( ; edge[0] > -1; edge += 3 )
  for (i=0; i<3; i++)
    vert = HexaEdges[edge[i]]; /* Calculate and store the three edge intersections */
      n0 = nodes[vert[0]]; n1 = nodes[vert[1]];
    t = (thresh - val1[n0]) / (val1[n1] - val1[n0]);
    tri_ptr[i] = add_intersection(n0, n1, t ); /* Save an index to the pt. */
  add_triangle( tri_ptr[0], tri_ptr[1], tri_ptr[2], zoneID ); /* Store the */
```

Efficient Searching

- With < 10% of the voxels contributing to the surface, it is a waste to look at every voxel.
- A voxel can be specified in terms of its interval, its minimum and maximum values.

Span Space
Span Space - Representing

- K-d Trees

Minimum

Split Min. axis
Split Max. axis