Contouring

Computer Animation and Visualisation – Lecture 13
Taku Komura
tkomura@inf.ed.ac.uk
Institute for Perception, Action & Behaviour
School of Informatics
Overview

- Contouring
  - Problem statement
  - Tracking
  - Marching squares
  - Ambiguity problems
  - Marching cubes
  - Dividing squares
Contouring

• Contours explicitly *construct* the **boundary** between regions with values

• Boundaries correspond to:
  – lines in 2D (known as isobars)
  – surfaces in 3D (known as isosurfaces)

  – of constant scalar value
Example: contours

- lines of constant pressure on a weather map (isobars)
- surfaces of constant density in medical scan (isosurface)
  - "iso" roughly means equal / similar / same as
Contouring

- **Input**: 2D or 3D grid with scalar values at the nodes
- **Output**: Contours (polylines, polygons) that connect the vertices with the same scalar value
2D Contour

• **Input Data:** 2D structured grid of scalar values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

• Difficult to visualise transitions in data
  - use **contour** at specific scalar value to highlight **transition**

• What is the contour of 5?
Methods of Contour Line Generation

• Approach 1: Tracking
  - find contour intersection with an edge
  - track it through the cell boundaries
    - if it enters a cell then it must exit via one of the boundaries
    - track until it connects back onto itself or exits dataset boundary
  - If it is known to be only one contour, stop
  - otherwise
    - Check every edge

• Approach 2: Marching Squares Algorithm
Marching Squares Algorithm

- **Focus**: intersection of contour and cell edges
  - how the contour passes through the cell
- **Assumption**: a contour can pass through a cell in only a finite number of ways
  - A vertex is inside contour if scalar value > contour
    - outside contour if scalar value < contour
  - 4 vertices, 2 states (in or out)
Marching Squares

- $2^4 = 16$ possible cases for each square
  - small number so just treat each one separately
MS Algorithm Overview

• **Main algorithm**

1. Select a cell
2. Calculate inside/outside state for each vertex
3. Look up topological state of cell in state table
   - determine which edge must be intersected (i.e. which of the 16 cases)
4. Calculate contour location for each intersected edge
5. Move (or march) onto next cell
   - until all cells are visited GOTO 2
MS Algorithm - notes

- Intersections for each cell must be merged to form a complete contour
  - cells processed independently
  - further "merging" computation required
  - disadvantage over tracking (continuous tracked contour)

- Easy to implement (also to extend to 3D)
- Easy to parallelise
MS : Dealing with ambiguity?

- Choice independent of other choices
  - either valid: both give continuous and closed contour
Example : Contour Line Generation

• 3 main steps for each cell
  - here using simplified summary model of cases
Step 1: classify vertices

- Decide whether each vertex is inside or outside contour

- No intersection.
- Contour intersects 1 edge
- Contour intersects 2 edges
- Ambiguous case.

Contour value = 5
Step 2 : identify cases

- Classify each cell as one of the cases

- No intersection.
- Contour intersects 1 edge.
- Contour intersects 2 edges.
- Ambiguous case.

Contour value = 5
Step 3 : interpolate contour intersections

- Determine the edges that are intersected
  - compute contour intersection with each of these edges
Ambiguous contour

- Finally: resolve any ambiguity
  - here choosing “join” (example only)
MS : Dealing with ambiguity ?
One solution

- Calculate the value at the middle of the square by interpolation
- Check if it is under or above the threshold value
- Choose the pattern that matches
Ambiguous contour

No intersection.

Contour intersects 1 edge

Contour intersects 2 edges

Ambiguous case.
Step 3 : interpolate contour intersections

- No intersection.
- Contour intersects 1 edge
- Contour intersects 2 edges
- Ambiguous case.

Split

No intersection.
Contour intersects 1 edge
Contour intersects 2 edges
Ambiguous case.
2D : Example contour

A slice through the head

A Quadric function.

(with colour mapping added)
3D surfaces : marching cubes

- Extension of Marching Squares to 3D
  - **data** : 3D regular grid of scalar values
  - **result** : 3D surface boundary instead of 2D line boundary
3D surfaces: marching cubes

- Extension of Marching Squares to 3D
  - **data**: 3D regular grid of scalar values
  - **result**: 3D surface boundary instead of 2D line boundary
  - 3D cube has 8 vertices → $2^8 = 256$ cases to consider
    - use symmetry to reduce to 15

- **Problem**: ambiguous cases
  - cannot simply choose arbitrarily as choice is determined by neighbours
  - poor choice may leave hole artifact in surface
Marching Cubes - cases

- Ambiguous cases
  - e.g., 3, 6, 10, 12, 13 – split or join?
Example of bad choices

- The dark dots are the interior
- There are edges which are not shared by both cubes
- Need to make sure there is no contradiction with the neighbors
Cracks eliminated
Adding more patterns

- Adding more patterns for 3, 6, 10, 12, 13 [Neilson '91]
- Compute the values at the middle of the faces and the cubes
- Selecting the pattern that matches
Other two possible triangulations

- Need to decide how the faces are intersected by the contours
Rendering Implicit Surfaces

The marching cubes algorithm is useful for rendering implicit surfaces where $F(x,y,z) = 0$

Inside : $F(x,y,z) > 0$
Outside: $F(x,y,z) < 0$
Marching Cubes by CUDA

- http://www.youtube.com/watch?v=Y5URxpX8q8U
Dividing Cubes Algorithm

• Marching cubes : Problem
  - Often produces more polygons than pixels for given rendering scale
  - Problem : causes high rendering overhead

• Solution : Dividing Cubes Algorithm
  - Draw points instead of polygons  (faster rendering)
  - Need  1: efficient method to find points on surface
    2: method to shade points
Example: 2D divided squares for 2D lines

Find pixels that intersect contour
- Subdivide them
2D “divided squares” for lines

Find pixels that intersect line
- **Subdivide them** (usually in 2x2)
- Repeat recursively
2D “divided squares” for lines

Find pixels that intersect line
- Subdivide them
- Repeat recursively
until screen resolution reached
• Fill in the pixel with the color of the line
Extension to 3D

- Find **voxels** which intersect **surface**
- Recursively subdivide the voxels that intersect the contour
  - Until the voxel fits within a pixel
- Calculate **mid-points of voxels**
- Calculate the color of the pixel by shading
Drawing divided cubes surfaces

- **surface normal** for lighting calculations
  - interpolate from voxel corner points

\[
\left( \frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right) \approx \left( \frac{F^{x+\Delta x} - F^{x-\Delta x}}{2\Delta x}, \frac{F^{y+\Delta y} - F^{y-\Delta y}}{2\Delta y}, \frac{F^{z+\Delta z} - F^{z-\Delta z}}{2\Delta z} \right)
\]

- **problem with camera zoom**
  - ideally dynamically re-calculate points
  - not always computationally possible
Dividing Cubes : Example

50,000 points
when sampling less than screen resolution structure of surface can be seen
Summary

• Contouring Theory
  - 2D: **Marching Squares Algorithm**
  - 3D: **Marching Cubes Algorithm** [Lorensen '87]
    - marching tetrahedra, ambiguity resolution
    - limited to regular structured grids
  - 3D Rendering: **Dividing Cubes Algorithm** [Cline '88]

Readings

• G.M. Nielson, B Hamann, “The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes”
• W.E. Lorensen, H.E. Cline, “Marching Cubes: A high resolution 3D surface construction algorithm”
• H.E. Cline, W.E. Lorensen and S. Ludke, “Two algorithms for the three-dimensional reconstruction of tomograms”