Volume Rendering

- Major visualisation tools: Volume Rendering, Animation
- Input: a scalar function, \( f(x,y,z) \), sampled in 3D
- Problem: to understand (visualise) it
- Value, \( f(x_p,y_p,z_p) \), characterises a point \((x_p,y_p,z_p)\) — it might represent a
  - density
    - X-ray tomography: tissue density
    - \(\gamma\)-ray tomography or seismic study: soil density
  - concentration
    - MRI scan: \(^1\)H, \(^{13}\)C, ... atom concentration
  - potential
    - calculated electric field
  - ...
Volume Rendering - Two strategies

- Traditionally we have differentiated
  - volume rendering
    - processes act directly on the data set.
  - geometric rendering
    - geometric primitives used to produce images
- Modern techniques and hardware
  - identical images from both techniques
- For instance,
  - ray-casting techniques (volume rendering using data set) => an iso-surface very similar to
  - one from rendering the geometric primitives resulting from a marching cubes calculation.

Volume Rendering - Two strategies (cont.)

- Distinction now blurred
- We tend to use the broader definition
  "Volume rendering is any method that operates on volumetric data to produce an image"
- Major application areas
  - Medical imaging
  - Various diagnostic modalities
    - CAT, MRI and PET
collect (usually) 3D stacks of images
  - Image values related to absorption coefficients
  - Volume data collected is usually very large

Colour mapping - Visualisation Example

- Map a scalar value to a colour
- Use a lookup table (usually)
- Mechanism
  - Use \( n \) colours
  - Store colours in table, indexed 0 to \( n - 1 \)
  - Value, \( s(x) \), to be visualised lies in \([ \text{min}, \text{max}]\)
- Algorithm:
  
  ```
  if s(x) < min then index = 0;
  if s(x) > max then index = n-1;
  else index = n*s(x) - min)/(max - min);
  s(x) = colour_table[index];
  ```

Colour Mapping

- Colour mapping is a one dimensional technique
  - we are mapping a scalar value to a single colour.
- However, the display is not limited to one dimension,
  - It can be 2-D or even 3-D:
**Colour Mapping**

- Key to colour mapping
  - *choice of lookup table*
- Table should
  - accentuate the important features, *while*
  - minimising the less important differences.
- Art or experience?
  - The design of a good mapping is as much an art as a science!

**Contouring**

- Natural extension to colour mapping.
- Construct the boundaries between the regions of different scalar values.
- Boundaries can be lines (2D) or surfaces (3D).
- Weather maps frequently use contours to show lines of constant value
  - isotherms for temperature,
  - isobars for pressure, etc.
- 3 dimensional contours are called iso-surfaces
  - later lectures!

**Contouring (cont.)**

**Example:**

Here is a structured grid

- Numbers in the cells represent values of some physical property, \( f(x) \), at the sample point
  (the cell vertex nearest the number)
- We want to show where \( f(x) = 5 \)
- Use linear interpolation
  - Simple, fast, usually sufficient
- Determine points for the contour on cell edges
  - Interpolate between two vertices
- Connect

**Contouring - 2D example**
Two basic strategies

- Contour tracking
- Marching squares

Contouring - Contour tracking

Contour entering a cell must exit
Detect an edge intersection
Track this contour as it crosses cell boundaries
Until it closes back on itself
or
Exits dataset boundary
If there is known to be only one contour, stop
otherwise
Check every edge

Marching Squares

- A divide and conquer technique
  - Previous algorithm evaluated every edge
- Try to be more efficient ...
- Basic assumption
  - there is only a finite number of ways a contour can pass through a cell
  - there is a limited number of topological states of a cell.
- Enumerate all of these states
  - place them in a table

Marching Squares

- Topological states depend on
  - the number of vertices (4),
  - the relationships between the vertex values and the contour of interest.
- Count the possibilities:
  - There are 4 vertices
  - value at each vertex value can be
    $\leq$ greater than (inside) or
    $\leq$ less than (outside) the contour,
  - so there are $4^4 = 16$ possible ways a contour can pass through a cell

Note: we are not interested (yet) in where the contour passes through the cell (the geometry) 
but how it passes through the cell (the topology).
Marching Squares

- Let's work out the possibilities:

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0110</td>
<td>0001</td>
<td>1011</td>
<td>0100</td>
<td>1001</td>
<td>1010</td>
<td>1111</td>
</tr>
</tbody>
</table>

- Point above contour (index bit = 1)

Marching Squares - Ambiguities

- Cases five (0101) and ten (1010) are ambiguous
- Two possible contours
- In 2D, choose either one

Marching Squares

- Algorithm
  - Select a cell
  - Calculate inside/outside state of each vertex
  - Generate topological state index from 'inside' bits
  - Look up topological state using index
    - Gives set of edges intersected by contour
  - Calculate contour location using interpolation for edges in the intersection set
    \[ r = \frac{C - f(0)}{f(1) - f(0)} \]
    - C is the desired contour value

- Notes:
  - Interpolate in the same direction for each edge
  - Otherwise round-off non-coincident points
  - Boundaries: duplicate edges & vertices may be created
  - Eliminate with coincident point-merging operation

Marching Squares - Ambiguities

- Two possible contours
- In 2D, choose either one

- Either acceptable
  - Resulting contour lines will be continuous or closed or end at dataset boundary
Contour Tracking vs Marching Squares

- **Marching Squares**
  - Easy to implement
    - Important in equivalent 3D algorithm
  - Creates disconnected line segments and points
    - Merging operation needed
- **Tracking**
  - Can generate a single polyline per contour line
    - Avoids the need to merge coincident points
  - Each edge must be checked for multiple contours

Marching Cubes

- Marching Squares extended into 3D
- Aim: Generate an iso-surface (contour)
- Fixed number of ways for a contour to pass through a cubic cell
- Voxel values at vertices

Marching Cubes

- Again contour passes inside or outside each vertex

Marching Cubes

- Again contour passes inside or outside each vertex
  - 8 vertices, so $2^8 = 256$ different topologies
  - Cube has a high degree of symmetry
    - Can reduce the number of distinct cases
    - reject cases which can be generated from others by symmetry operations
  - What are the symmetry operations for a cube?
Marching Cubes - Distinct topologies

- Also ..
- Complementary symmetry
  - Inside / Outside choice is arbitrary
  - Half number of cases by complementing inside / outside choice
  - Location of surface doesn’t change,
  - Only direction of surface normal

- So 256 cases reduce to ...

Marching Cubes - Surface Normals

- Surfaces represented as sets of triangles
- Surface normals need to be determined for shading
- Requires consistent vertex ordering correct normals

- Only 15!

Marching Cubes - Surface Normals

- How to determine surface normals ...
  Can use geometry of triangles used to approximate surface
  - Rendering hardware can do this!
  Determine vertex normals at each triangle vertex
  - Use normals at cube corners
  - Derived from gradient of the voxel values
  - Interpolate to find triangle vertex normals
  - More complex
  - Potentially more accurate
Marching Cubes - Efficiency

- Large amounts of data
  - Design must take into account the limitations on the computing hardware
  - Typically memory - still a problem nowadays?
- Efficiency issues
  - Implementation should make use of coherency – no contour-edge intersections should be calculated twice!
    - Note: intersection of contour with an edge is relevant to three cubes (ignoring cubes on the limits of the volume data)
  - Minimize memory use see later

Marching Cubes - Algorithm

- Full algorithm in Watt and Watt (and elsewhere)
- Assumptions
  - Input data is equi-spaced and planar.
  - Iso-surface required is derived from a set of closed contours on each plane
  - Since the data is volumetric, each data element is labelled by the indices (i,j,k).
  - These indices map to some spatial measure, if we know
    - the starting values of (x_0,y_0,z_0)
    - the spacing between voxels in each direction: \( \delta x, \delta y, \delta z \)
    - then the position \( (x,y,z) \) in the march space of an element \( (i,j,k) \) is given by
      \[
      x = x_0 + i \delta x
      \]
      same in the y and z directions

Marching Cubes - Algorithm

- Memory use is minimized
  - the marching cubes algorithm is based on relationships between nearest neighbours.
  - Hence at most only two slices need to be retained in memory at any one time.

Marching Cubes - Algorithm

- Memory use is minimized
  - the marching cubes algorithm is based on relationships between nearest neighbours.
  - Hence at most only two slices need to be retained in memory at any one time.

- March ordering:
  - from element \( (i,j,k) \) to \( (i,j+1,k) \) (march pixel by pixel incrementing \( j \)), then
  - row by row (incrementing \( i \)) and
  - finally slice by slice (incrementing \( k \)).

1 - along each column
2 - row by row
3 - slice by slice
Marching Cubes - Algorithm
• Coherency is exploited by storing relevant edge intersections for later reuse.

Marching Cubes - Cell Topology
• Need a consistent edge & vertex numbering scheme, eg

Marching Cubes - Storing the data
A model contains both vertex data and facet data
• Vertex data (i.e. intersections of contour with marching cubes' edges) are stored in a vertex table
• Each facet (of the surface) consists of a list of vertex indices of vertices constituting the facet
  • Order of vertices in the facet is important, as it defines the direction of the surface normal!
  • Buffers are needed to retain data until the next slice is complete
    • then they may be deleted or re-used
    • some book-keeping challenges for a programmer!

/* C is contour value */
/* p[k], k=0,7 contains vertex values */
index = 0;
for ( k=0; k<8; k++ ) {
  if ( p[k] > C ) index = index | (1<<k);
}

/* topology is a reference to a suitable structure which contains intersecting edge list */
 topology = top_table[index];
Effect of grid size

- Grid Size vs Facets generated

<table>
<thead>
<tr>
<th>Grid size</th>
<th>Facets</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>220</td>
</tr>
<tr>
<td>2</td>
<td>1700</td>
</tr>
<tr>
<td>0.5</td>
<td>6800</td>
</tr>
<tr>
<td>0.5</td>
<td>27000</td>
</tr>
</tbody>
</table>

Further reading