Scalar Algorithms -- surfaces

✓ Color Mapping
✓ Slicing and Clipping
✓ Contouring / iso-surface
Sources of Scalar Data

✓ Sensors
  – CT/MRI
  – Surface Scanners: laser/range scans

✓ Simulations
  – CFD/FEM

✓ Mathematical
  – Implicit functions
Surface Graphics

- Employs surface primitives (polygons, B-splines, NURBS) for scene representation
- photo-realistic
- “hollow” objects (not true 3D)
2D Data

Examples: Medical scans, GIS data

✓ Color mapping
✓ Contour extraction
✓ Height field
Color Mapping

- Color lookup table using the scalar attributes as indices into the lookup table
- Widely used in slicing and volume rendering.

\[
\begin{cases}
    s_i < \text{min}, & i = 0 \\
    s_i > \text{max}, & i = n - 1 \\
    \text{otherwise}, & i = n \cdot (s_i - \text{min}) / (\text{max} - \text{min})
\end{cases}
\]
Color Mapping (2)

✓ **Transfer function**: a more general form of lookup table. A function mapping from scalar values to color values.

![Graphs of Red, Green, and Blue channels for color mapping](image)
VTK colormap examples

```cpp
plane = new vtkStructuredGridGeometryFilter;
plane->SetInput(datafile);
plane->SetExtent(1,100,1,100,7,7);
lut = new vtkLookupTable;
planeMapper = new vtkPolyDataMapper;
planeMapper->SetLookupTable(lut);
planeMapper->SetInput(plane->GetOutput());
planeMapper->SetScalarRange(0.197813, 0.710419);
planeActor = new vtkActor;
planeActor->SetMapper(planeMapper);
```
Greyscale Table

```cpp
lut->SetHueRange(0,0);
lut->SetSaturationRange(0,0);
lut->SetValueRange(0.2,1.0);
```
Rainbow - blue to red

lut->SetHueRange(0.667, 0.0);
Rainbow - red to blue

```cpp
lut->SetHueRange(0.0, 0.667);
```
Enhancing Contrast

```c
lut->SetNumberOfColors(64);
lut->Build();
for(i=0;i<16;i++) {
    lut->SetTableValue(i*4,red);
    lut->SetTableValue(i*4+1,green);
    lut->SetTableValue(i*4+2,blue);
    lut->SetTableValue(i*4+3,black);
}
lut->SetTableValue(0,coral);      lut->SetTableValue(1,black);
lut->SetTableValue(2,peacock);    lut->SetTableValue(3,black);
lut->SetTableValue(4,orchid);     lut->SetTableValue(5,black);
lut->SetTableValue(6,cyan);       lut->SetTableValue(7,black);
lut->SetTableValue(8,mint);       lut->SetTableValue(9,black);
lut->SetTableValue(10,tomato);    lut->SetTableValue(11,black);
lut->SetTableValue(12,sea_green); lut->SetTableValue(13,black);
lut->SetTableValue(14,plum);      lut->SetTableValue(15,black);
```
Contour Extraction

✓ A contour curve consists of points with the same attribute value
  - pixel intensity
  - height
  - temperature
// Read in the data.

v16 = new vtkVolume16Reader;
v16->SetDataDimensions(128,128);
v16->GetOutput()->SetOrigin(0.0,0.0,0.0);
v16->SetFilePrefix("../../data/headsq/half");
v16->SetImageRange(45,45);
v16->SetDataAspectRatio(1.6,1.6,1.5);

iso = new vtkContourFilter;
iso->SetInput(v16->GetOutput());
iso->GenerateValues(12,500,1150);
// Map the contour output to polygons, and create the actor!
isoMapper = new vtkPolyMapper;
isoMapper->SetInput(iso->GetOutput());
isoActor = new vtkActor;
isoActor->SetMapper(isoMapper);
isoActor->GetProperty()->SetColor(black);
Height Fields

✓ GIS data: Terrain images
✓ Using height values to represent other attributes (e.g. population).
✓ Others: Gene Terrain, etc.
Definition: Slice Operator

✓ Intersect a cell with a region: reducing the topological dimension of the cell by 1.

✓ \( \text{Slice}(C_d:F,V) \rightarrow C_{d-1} \)
  - \( C_d \): Cell of topological dimension \( d \)
  - \( F \): The scalar field defined at the vertices of the cells.
  - \( V \): The value defining the zero value of the scalar field.
Triangle Slicing
Slicing Triangles with Planes

\[ F(x,y,z) = n_x x + n_y y + n_z z - d = 0 \]
Conventional Slicing (contouring)
Slicing Hexahedra with Planes
VTK Example: 3D Contouring of Quadric

\[ F(x,y,z) = a_0 x^2 + a_1 y^2 + a_2 z^2 + a_3 xy + a_4 yz + a_5 xz + a_6 x + a_7 y + a_8 z + a_9 \]
quadric = new vtkQuadric;
quadric->SetCoefficients(1, 2, 3, 0, 1, 0, 0, 0, 0, 0);

sample = new vtkSampleFunction;
sample->SetSampleDimensions(40, 40, 40);
sample->SetImplicitFunction(quadric);

contour = new vtkContourFilter;
contour->SetInput(sample->GetOutput());
range[0] = 1.0; range[1] = 6.0;
contour->GenerateValues(3, range);

contourMapper = new vtkPolyDataMapper;
contourMapper->SetInput(contour->GetOutput());
contourMapper->SetScalarRange(0, 7);
contourActor = new vtkActor;
contourActor->SetMapper(contourMapper);
VTK Example: 3D Contouring (Iso-Surface) of CT Data

// Read in the data.

v16 = new vtkVolume16Reader;
v16->SetDataDimensions(128,128);
v16->GetOutput()->SetOrigin(0.0,0.0,0.0);
v16->SetFilePrefix("../../data/headsq/half");
v16->SetImageRange(1,93);
v16->SetDataAspectRatio(1.6,1.6,1.5);

iso = new vtkMarchingCubes;
iso->SetInput(v16->GetOutput());
iso->SetValue(0,1150);
// Map to polys and create the actor.
isoMapper = new vtkPolyDataMapper;
isoMapper->SetInput(iso->GetOutput());
isoActor = new vtkActor;
isoActor->SetMapper(isoMapper);
isoActor->GetProperty()->SetColor(antique_white);
VTK Example: 3D Contouring (Iso-Surface) of Iron Protein Molecule (electron potential)

```cpp
reader = new vtkStructuredPointsReader;
reader->SetFilename("../../data/ironProt.vtk");

iso = new vtkContourFilter;
iso->SetInput(reader->GetOutput());
iso->SetValue(0, 128);

// Clean up duplicate points.
clean = new vtkCleanPolyData;
clean->SetInput(iso->GetOutput());
```
normals = new vtkPolyNormals;
normals->SetInput(clean->GetOutput());
normals->SetFeatureAngle(45);

isoMapper = new vtkPolyMapper;
isoMapper->SetInput(normals->GetOutput());
isoActor = new vtkActor;
isoActor->SetMapper(isoMapper);
isoActor->GetProperty()->SetColor(bisque);
Definition: Clip operator

✓ Determining which part of a cell lies within a clip region, maintaining the topological dimension of the cell.

✓ \( \text{Clip}(C_d:F,V) \rightarrow C_d \)
Triangle Clipping
Clipping with Quadrics
Tissue Lens - Clip Isosurface

Clip Triangle
Cells on
Isosurface
(Value = 600)

\[ f(x,y,z) = x^2 + y^2 + z^2 - r^2 \]

Value = \( r^2 \)
Tissue Lens - Clip Lens Geometry

Clip Quad Cells on a Sphere

\[ f(x,y,z) = \text{Implicit Volume} \]

Value = 600
Spherical Tissue Lens
Cubical Tissue Lens
Isosurface Extraction

✓ Contouring in 3D
  - closed contours
  - continuous
  - determined by iso-values

✓ several methods
  - marching cubes
  - dividing cubes
  - surface tracking
  - Sorting: span space, etc

Iso-value=5
Marching Cubes

Cell consists of 8 voxel values: (i+[01], j+[01], k+[01])

1. Consider a Cell
2. Classify each vertex as inside or outside
3. Build an index
4. Get edge list from table[index]
5. Interpolate the edge location
6. Go to next cell
Create a Cube

Consider a Cube defined by eight data values:

- $(i,j,k)$
- $(i+1,j,k)$
- $(i,j+1,k)$
- $(i,j,k+1)$
- $(i,j+1,k+1)$
- $(i+1,j+1,k+1)$
- $(i+1,j,k+1)$
- $(i+1,j+1,k)$

Diagram of a cube with vertices labeled accordingly.
Classify Each Voxel

✓ Classify each voxel according to whether it lies outside the surface (value > iso-surface value) inside the surface (value <= iso-surface value)

![Diagram showing classification of voxels with different isosurface values]

- Iso=7
  - =inside
  - =outside

- Iso=9
Build An Index

✓ Use the binary labeling of each voxel to create an index

✓ all 256 cases can be derived from 15 base cases

Index:

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
</tr>
</thead>
</table>

- inside = 1
- outside = 0
Lookup Edge List

✓ For a given index, access an array storing a list of edges

✓ Index = 10110001
✓ triangle 1 = e4, e7, e11
✓ triangle 2 = e1, e7, e4
✓ triangle 3 = e1, e6, e7
✓ triangle 4 = e1, e10, e6
Interp. Triangle Vertex

✓ For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values

\[
x = i + \left( \frac{T - v[i]}{v[i+1] - v[i]} \right)
\]

\begin{align*}
T &= 5 \\
T &= 8
\end{align*}
Compute Normals

 ✓ Calculate the normal at each cube vertex

\[ G_x = v_{i+1,j,k} - v_{i-1,j,k} \]
\[ G_y = v_{i,j+1,k} - v_{i,j-1,k} \]
\[ G_z = v_{i,j,k+1} - v_{i,j,k-1} \]

 ✓ Use linear interpolation to compute the polygon vertex normal

 ✓ Alternatively, the normals of the triangle vertices may be directly computed by finite difference.
Ambiguous Cases

✓ Ambiguous cases: 3, 6, 7, 10, 12, 13
✓ Adjacent vertices have different states and diagonal vertices have same state
✓ Resolution: decide for one case
  - Marching tetrahedra
  - Complementary cases
  - Asymptotic decider
Ambiguous Cases : Holes

(a) Break contour

(b) Join contour

Case 3

Case 6c
Marching tetrahedra

✓ Subdivide voxels into tetrahedra with compatible neighborhood
Complementary cases

Case 3c

Case 6c

Case 7c

Case 10c

Case 12c

Case 13c
Asymptotic Decider

- Assume bilinear interpolation within a face
- hence iso-surface is a hyperbola
- compute the point p where the asymptotes meet
- sign of $S(p)$ decides the connectedness
Dividing Cubes/Squares

- Generating and rendering Points (faster to draw).
- Principle - Divide a square to pixel sized cells and draw points in cells where the contour passes
- Work with both structured and unstructured grids
- Need dense points, scaling is a problem.

Iso-value=5
Dividing Cubes

1. Create a cube
2. Classify each voxel
3. Subdivide surface cubes to image resolution
4. Calculate normals
5. Output a point (the center point) for each surface cube
Step 3: Subdivide

✓ Subdivide each cube such that the volume resolution is higher than the image resolution. For example, when a $128^3$ volume is rendered to a $512^2$ image, surface cubes will be divided to 4x4x4 smaller cubes. ($n = \text{voxel size} / \text{pixel size}$)

classify

subdivide

output points
Uniform subdivision

Contour line

Find intersecting pixel
Subdivide pixel
Generate center points

Contour surface

Find intersecting voxel (single voxel shown)
Subdivide voxel
Generate points
Recursive (adaptive) subdivision
Surface Tracking

Given function $S(t, x, y, z)$ that returns 1 if $(x, y, z)$ is on the surface $t$ and 0 otherwise.

Needed: a seed voxel $p$ that is on the surface.
Surface Tracking

Let Q be a queue of voxels.

Push (enqueue) p onto Q
Flag p as “visited”
While Q is not empty do
    Pop (dequeue) q from Q and output it.
    For each voxel v in the neighborhood of q do
        If S(t, v) = 1 then
            If v was not visited then
                Push v onto Q
                Flag it as “visited”

end {while}
Efficient Tracking - Itoh & Koyamada

- An efficient way to find seed cells
- Pre-processing: Locate extrema and boundaries
- At least one local-min/max point inside a closed iso-surface (contour 1)
- If an iso-surface is not closed, it has intersections with the boundary of a volume (contour 2).
Extract local-minimum or local-maximum points in a volume.

Generate a graph which connects all the extrema points.

For each cell, all nodes except the maximum are marked “not maximum” and all nodes except the minimum are marked “not minimum”.

After all cells are processed, only the nodes that don’t have both “not maximum” and “not minimum” are selected as extremum points.
Intervening cells are inserted into a cell list.

Cells which touch the boundary of a volume are registered into another cell list. At least one cell is necessarily intersected by an iso-surface.
Check cells registered in the cell lists generated in the pre-process

When a cell intersected by an iso-surface is found, adjacent intersected cells are traversed recursively and intersections are approximated as polygons
Span Space

✓ Create two lists
✓ store (in order) the minima of each cell into the first list
✓ store (in order) the maxima of each cell into the second list
✓ Span space point: (xmin, xmax), and each cell corresponds to one point.
✓ By H. Shen and Chris Johnson (U of Utah)
Span Space

✓ **Given**: iso-value v
✓ different algorithms explore this span-space with different efficiencies
✓ Basically a range search problem
✓ **Given:** iso-value $v$ and max range $r$

✓ gracefully update to new value $nv$

✓ **Active list:** All cells with minimum values between $v-r$ and $v$, and maximum value between $v$ and $v+r$

✓ **Update active list:** Adding and purging only the difference regions.
Span Space - kd-tree

✓ Build efficient search tree
✓ Kd-tree: multi-dimensional binary search tree:
  Quicksort partitioning for “min”, and next level for “max, etc.
✓ Search:
  if (root.min < v) {
    if (root.max > v)
      construct polygon;
    search (v, root.right)
  }
  search (v, root.left);