CSE 564
Visualization & Visual Analytics

Visualizing Volumetric Data

Klaus Mueller

Computer Science Department
Stony Brook University
<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intro, schedule, and logistics</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Applications of visual analytics, basic tasks, data types</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Introduction to D3, basic vis techniques for non-spatial data</td>
<td>Project #1 out</td>
</tr>
<tr>
<td>4</td>
<td>Data assimilation and preparation</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Data reduction and notion of similarity and distance</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Visual perception and cognition</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Visual design and aesthetics</td>
<td>Project #1 due</td>
</tr>
<tr>
<td>8</td>
<td>Dimension reduction</td>
<td>Project #2 out</td>
</tr>
<tr>
<td>9</td>
<td>Data mining techniques: clusters, text, patterns</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Cluster analysis: numerical data</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Cluster analysis: categorical data</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Spatial data origins: medical imaging, scientific simulation</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td><strong>Techniques to visualize spatial data: volume visualization</strong></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Intro to GPU programming</td>
<td>Project #2 due / Project #3 out</td>
</tr>
<tr>
<td>15</td>
<td>Techniques to visualize spatial data: flow visualization</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Midterm #1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>High-dimensional data, dimensionality reduction</td>
<td>Project #3 due</td>
</tr>
<tr>
<td>18</td>
<td>Big data: data reduction, summarization</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Correlation and causal modeling</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Principles of interaction</td>
<td>Final project proposal due</td>
</tr>
<tr>
<td>21</td>
<td>Visual analytics and the visual sense making process</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Evaluation and user studies</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Visualization of time-varying, time-series, streaming data</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Visualization of graph data</td>
<td>Final Project preliminary report due</td>
</tr>
<tr>
<td>25</td>
<td>Visualization of text data</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Midterm #2</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Data journalism</td>
<td>Final Project slides and final report due</td>
</tr>
</tbody>
</table>
Volume Data Generation

Often obtained by scanning

- for example, X-ray CT
Volume Data – 2D Slice View
Volume Data – 3D Rendered View

aneurism
broken jaw

Which do you prefer: 2D or 3D

carotid arteries
Raycasting Concept

Eye → Image Plane

Data Set

Numerical Integration

Resampling
SAMPLING ALONG THE RAY

Estimate sample values via interpolation
$f_v = f_1(1 - p)(1 - q)(1 - r) + f_2(p)(1 - q)(1 - r) + f_3(p)q(1 - r) + f_4(1 - p)(q)(1 - r) + f_5(1 - p)(1 - q)(r) + f_6(p)(1 - q)(r) + f_7(p)q(r) + f_8(1 - p)(q)(r)$
We learned about RGB

There is one more channel – opacity (A)

- gives RGBA color
- opacity (A) = 1 – transparency (T)
- range [0.0 ... 1.0]

Opacity (A) multiplied by RGB creates a weighting effect
Opacity and Color Blending

\[ C_{\text{mix}} = C_{\text{front}} A_{\text{front}} + (1 - A_{\text{front}}) C_{\text{back}} A_{\text{back}} \]

\[ C_{\text{mix}} = C_B A_B + (1 - A_B) C_R A_R \]

\[
T_R = 0.00, \ A_R = 1.00
\]

\[
C = R \cdot 0.75 + B \cdot 0.25
\]

\[
T_B = 0.75, \ A_B = 0.25
\]

\[
C = R \cdot 0.50 + B \cdot 0.50
\]

\[
T_B = 0.50, \ A_B = 0.50
\]

\[
C = R \cdot 0.25 + B \cdot 0.75
\]

\[
T_B = 0.25, \ A_B = 0.75
\]

\[
C = R \cdot 0.00 + B \cdot 1.00
\]

\[
T_B = 0.00, \ A_B = 1.00
\]
Compositing – Merging the Samples

Back-to-front rendering

\[ C'_i = C_i A_i + (1 - A_i) C'_{i-1} \]

Front-to-back rendering

\[ C'_i = C'_{i-1} + (1 - A'_{i-1}) C_i A_i \]
\[ A'_i = A'_{i-1} + (1 - A'_{i-1}) A_i \]

A: Opacity = 1 - Transparency

C: Color
A ray is specified by:
- eye position (Eye)
- screen pixel location \( P_{i,j} \)
\[ r_{i,j} = \frac{P_{i,j} - \text{Eye}}{|P_{i,j} - \text{Eye}|} \]

A point \( P \) on a ray is given by:
\[ P = \text{Eye} + t \cdot r_{i,j} \]

\( t \): parametric variable

Spacing of pixels on image plane:
\[ \Delta i = \frac{W}{N_i - 1} \quad \Delta j = \frac{H}{N_j - 1} \]

\( N_i, N_j \): image dims. in pixels

Image-order projection:
- scan the image row by row, column by column:
\[ P_{i,j} = P_{0,0} + i \cdot v \cdot \Delta j + j \cdot u \cdot \Delta i \]

- \( P_{i,j} \): Location of image pixel \((i, j)\) in world space

- \( P_{0,0} \): image (=screen) origin in world space

- \( u, v, n \): orthonormal image plane vectors \((n = v \times u)\)
**Volume Rendering Modes**

- **X-ray:**
  rays sum volume contributions along their linear paths

- **Iso-surface:**
  rays look for the object surfaces, defined by a certain volume value

- **Maximum Intensity Projection (MIP):**
  a pixel value stores the largest volume value along its ray

- **Full volume rendering:**
  rays *composite* volume contributions along their linear paths
**Practical Implementation**

- Everything handled in the fragment shader
- Procedural ray / bounding box intersection
- Ray is given by camera position and volume entry position
- Exit criterion needed
- Pro: simple and self-contained
- Con: full load on the fragment shader
GPU PROGRAM

- Rasterize front faces of volume bounding box
- Texcoords are volume position in [0,1]
- Subtract camera position
- Repeatedly check for exit of bounding box

// Cg fragment shader code for single-pass ray casting
float4 main (VS_OUTPUT IN, float4 TexCoord0 : TEXCOORD0,
uniform sampler3D SamplerDataVolume,
uniform sampler1D SamplerTransferFunction,
uniform float3 camera,
uniform float stepsize,
uniform float3 volExtentMin,
uniform float3 volExtentMax ) : COLOR
{
    float4 value;
    float scalar;
    // Initialize accumulated color and opacity
    float4 dst = float4(0,0,0,0);
    // Determine volume entry position
    float3 position = TexCoord0.xyz;
    // Compute ray direction
    float3 direction = TexCoord0.xyz - camera;
    direction = normalize(direction);
    // Loop for ray traversal
    for (int i = 0; i < 200; i++) // Some large number
    {
        // Data access to scalar value in 3D volume texture
        value = tex3D(SamplerDataVolume, position);
        scalar = value.a;
        // Apply transfer function
        float4 src = tex1D(SamplerTransferFunction, scalar);
        // Front-to-back compositing
        dst = (1.0-dst.a) + src + dst;
        // Advance ray position along ray direction
        position = position + direction * stepsize;
        // Ray termination: Test if outside volume ...
        float3 temp1 = sign(position - volExtentMin);
        float3 temp2 = sign(volExtentMax - position);
        float inside = dot(temp1, temp2);
        // ... and exit loop
        if (inside < 3.0)
            break;
    }
    return dst;
}
Why is front-to-back rendering better?

- early ray termination – terminate a ray when $A > 0.90$

- empty-space skipping – jump across empty space quickly
ISO-SURFACE RENDERING

• A closed surface separates ‘outside’ from ‘inside’ (Jordan theorem)
• In iso-surface rendering we say that all voxels with values > some threshold are ‘inside’, and the others are ‘outside’
• The boundary between ‘outside’ and ‘inside’ is the iso-surface
• All voxels near the iso-surface have a value close to the iso-threshold or iso-value
• Example:

  cross-section of a smooth sphere

  iso-boundary

  iso-value = 50 will render a large sphere

  iso-value = 200 will render a small sphere
Iso-Surface Rendering

iso-value = 30  iso-value = 80  iso-value = 200
To render an iso-surface we cast the rays as usual...

but we stop, once we have interpolated a value \textit{iso-threshold}

\begin{itemize}
\item We would like to illuminate (shade) the iso-surface based on its orientation to the light source
\item Recall that we need a normal vector for shading
\item The normal vector $\mathbf{N}$ is the local gradient, normalized
\end{itemize}
THE GRADIENT VECTOR

- The gradient vector \( \mathbf{g} = (g_x, g_y, g_z)^T \) at the sample position \((x, y, z)\) is usually computed via central-differencing (for example, \(g_x\) is the volume density gradient in the \(x\)-direction):

\[
\begin{align*}
g_x &= \frac{f(x - 1, y, z) - f(x + 1, y, z)}{2} \\
g_y &= \frac{f(x, y - 1, z) - f(x, y + 1, z)}{2} \\
g_z &= \frac{f(x, y, z - 1) - f(x, y, z + 1)}{2}
\end{align*}
\]

Diagram showing the x and y component of the gradient vector for the smooth sphere.

- 2D central difference mask:

\[
\begin{array}{ccc}
0 & 0.5 & 0 \\
-0.5 & 0 & 0.5 \\
0 & -0.5 & 0
\end{array}
\]

- Voxels:
  - Voxel value = iso-threshold
  - Voxel value < iso-threshold
  - Extra sample points interpolated to estimate gradient
SHADING THE ISO-SURFACE

- The normal vector is the *normalized* gradient vector $g$
  \[ N = g / |g| \] (normal vector always has unit length)
- Once the normal vector has been calculated we shade the iso-surface at the sample point
- The color so obtained is then written to the pixel that is due to the ray

The color is calculated with the standard shading equation:

\[
C = C_{\text{obj}} \left( k_a I_A + k_d I_L \cdot N \cdot L \right) + k_s I_L (H \cdot N)^{ns}
\]

$C_{\text{obj}}$ is obtained by indexing the color transfer function with the interpolated sample value
When hitting a surface set $A < 1.0$
- ray marches on
- inner structures can be seen
During Classification the user defines the \textit{\textquote{Look}} of the data.

- Which parts are transparent?
- Which parts have which color?
During Classification the user defines the "Look" of the data.
- Which parts are transparent?
- Which parts have which color?
The user defines a Transfer function.
Classification
Classification
Classification
Classification
Real-Time update of the transfer function necessary!!!
Transfer Functions: Multi-Dimensional

Boundaries in volume create arches in \((\text{value}, \text{gradient})\) domain [Kindlmann 98]

Arches guide placement of opacity to emphasize material interfaces [Kniss 01]
Transfer Functions: Multi-Dimensional

- Boundaries can be described in terms of:
  - maximum in 1st derivative
  - zero-crossing in 2nd derivative
- Semi-automatic classification possible in clean data
Transfer Functions: Multi-Dimensional

Dual-domain interaction:

[Kniss 01]

Make features opaque by pointing at them

Actions in spatial domain

New transfer function

New Rendering

Changes to transfer function
Multi-Dimensional Transfer Functions
Multi-Dimensional Transfer Functions
Transfer Functions: Clinical Practice

A single slider bar is most appreciated [Rezk-Salama Vis06]

Enables doctors to quickly fine-tune the transfer function for specific objects

• works since in CT usually only small deviations exist
• but these require complex interactions in the transfer function domain
Parameter Mapping Approach (1)

Typical transfer function parameterization:

Datasets typically only deviate modestly from this
• but in complex ways
• meaning, lots of tweaking is required

[Rezk-Salama Vis06]
Parameter Mapping Approach (2)

We can learn these deviations by observing a few datasets

- encode the parameters into an N-D vector
- find the principal component of the vectors (the main Eigenvector)
- project all other vectors onto this Eigenvector
- the min and max then represent the min and max of the slider

[Rezk-Salama Vis06]