This part of the course deals with

„An interdisciplinary branch of science, primarily concerned with the visualization of three dimensional phenomena, such as architectural, meteorological, medical, biological systems. The emphasis is on [realistic] rendering of volumes, surfaces, [illumination sources], and so forth, perhaps with a dynamic (time) component.“
**Course Overview**

1. Introduction and Basics
2. The Data to be Visualized
3. Scalar Visualization Techniques
4. Vector Visualization Techniques
5. Color Mapping
6. Volume Rendering Theory
7. Volume Rendering Praxis
8. Perception and Cognition
9. Volumetric Haptics
10. Medical Visualization Challenges
11. (Reserved)
**LITERATURE**

- Additional books will be recommended during the course.

---

**THE SciVis GROUP AT LiU**

**Staff**
- Mark Eric Dieckmann
- Björn Gudmundsson
- Karljohan Lundin Palmerius
- Claes Lundström
- Timo Ropinski
- Anders Ynnerman

**PhDs & Research Engineers**
- Alexander Bock
- Daniel Jönsson
- Sathish Kottravel
- Stefan Lindholm
- Tan Khoa Nguyen
- Erik Sundén
**SciVis Research Areas**
- Volume Rendering
- Medical Visualization
- Haptics
- Visualization in other Domains

**SciVis Website**

[Website](http://scivis.itn.liu.se)
WORKING WITH THE GROUP

- Student co-assistant opportunities
- Master Thesis projects in interesting environments
- PhD positions

CURRENT RESEARCH

TNM093 - Practical Data Visualization
Introduction
APPLICATION AREAS

GOOGLE EARTH
**Visible Human Project**

**Humpback Whale Behavior**

[Ware et al., IEEE CG&A 2006]
**DISCIPLINES IN SCIENTIFIC VISUALIZATION**

- Data Domain-based visualization
  - Medical data ⇒ Volume Visualization
  - Flow data ⇒ Flow Visualization
  - GIS data ⇒ Geovisualization

- Other fields
  - Microscopic data (molecular physics),
  - Macroscopic data (astronomy)
  - Extremely large data sets
  - etc. ...

**MODERN APPLICATION AREAS**

- Medical visualization
- Geovisualization
  - 3D GIS
  - Oil and gas industry
- Industrial design
- Biological visualization
- ...

[Baumann 2000]
**MEDICAL VISUALIZATION**

- Image-based diagnosis
- Pre-operative planning
- Illustrative techniques for presentation and communication

**GEOVISUALIZATION - CITY VISUALIZATION**

- Different application areas
  - Tourism
  - Location-based services
  - Planning & presentation
  - ...
GEOVISUALIZATION - SEISMIC VISUALIZATION

- Problem
  - Find optimal location for well drilling (injection and extraction)

- Approach
  - Leverage human intuition
  - Couple simulation and visualization

VISUALIZATION TOOLKITS
VTK – THE VISUALIZATION TOOLKIT 1/2

- Open-source, freely available software system for 3D computer graphics, image processing, and visualization
- VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python
  - scalar, vector, tensor, texture, and volumetric methods
  - polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation
  - information visualization
  - 3D interaction widgets

[www.vtk.org]

VTK – THE VISUALIZATION TOOLKIT 2/2

[Image of VTK interface with diagrams and code snippets]

[www.vtk.org]
**AMIRA**

- Commercial toolkit for visualization in life sciences

[www.amiravis.com]

---

**MEVIS LAB**

[www.mevislab.de]
DATA STRUCTURES
**DATASOURCES**

- Two main sources for data
  - Real world (Measurement) collected through measurements or observations
  - Theoretical world (Simulation) computed on the basis of mathematical models

- Size of data depends on
  - Number of samples
  - Number of parameters per sample
  - Number of possible values per parameter

---

**MEASUREMENT & SIMULATION**

![Measurement](image1)

![Simulation](image2)
**DATA SPACE**

- Data space is the space in which the data is measured or simulated
- The data is acquired for different *observation points*
  - e.g., sensors in a sensor network

**DATA SPACE - DIMENSIONALITY**

- Dimensionality (1D, 2D, 3D,...) determines the number of *independent variables*
  - Discrete vs. continuous
    (4D: running clock vs. epoche-based)
- Scientific visualization: Inherent spatial domain
  - 2D/3D data space given
  - Examples: medical data, flow simulation data, GIS-data, ...
- Information visualization: No inherent spatial reference
  - Abstract data, *spatializing* through visualization
  - Examples: data bases, stocks, ...
**Data Space - Influence**

- Three different areas of influence exist for observation points
  - Punctual influence: just at the location of the observation point
  - Local influence: in its vicinity
  - Global influence: for the entire data space
- Punctual and local influence can be visualized spatially
- Local influence requires interpolation

**Data Space - Connectivity**

- Scattered data vs. grid structures
**DATA CHARACTERISTICS**

- **Data type**
  - Characteristic of a data value (precision & semantic)

- **Data dimensionality**
  - Scalar, multivariate / multimodal

- **Data domain**
  - Range of possible values

---

**CHARACTERISTICS - DATA DIMENSIONALITY**

- Specifies the number of components (parameters/attributes)
  - **Scalar**
    - Each data element has a numeric expression
    - Example: topography of terrain, expressed as 2D field containing elevations
  - **Vector**
    - Each data element is considered a straight directed line with a certain length (magnitude) in nD space
    - Example: direction of particle flow in a channel
  - **Tensor**
    - Each data element is expressed as a (transformation) matrix
    - Example: higher-order tensor fields
### CHARACTERISTICS - DATA DIMENSIONALITY

- The number of *dependent variables* at each observation point

<table>
<thead>
<tr>
<th>Data Dimensionality</th>
<th>1D</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = f(x)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td></td>
<td>Flow simulation v(x)</td>
<td></td>
</tr>
<tr>
<td>3D</td>
<td>Medical scans (CT, MRI, ...) i(x)</td>
<td>Multimodal scans (PET/CT) i1(x), i2(x)</td>
<td>DTI scans Flow simulation v(x)</td>
</tr>
</tbody>
</table>

### CHARACTERISTICS - DATA DIMENSIONALITY

<table>
<thead>
<tr>
<th>Data Space / Data Dimensionality</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>N_1 x R_1</td>
<td>value series</td>
<td>bar chart</td>
</tr>
<tr>
<td>R_1 x R_1</td>
<td>function</td>
<td>graph</td>
</tr>
<tr>
<td>R_2 x R_1</td>
<td>function over R_2</td>
<td>2D height map, contour lines, false color map</td>
</tr>
<tr>
<td>N_2 x R_2</td>
<td>2D vector field</td>
<td>hedgehog plot, LIC</td>
</tr>
<tr>
<td>R_3 x R_n</td>
<td>3D intensities</td>
<td>volume rendering, iso-surface rendering</td>
</tr>
<tr>
<td>(N_1)^+ x R^n</td>
<td>tuple set</td>
<td>parallel coordinates, glyphs</td>
</tr>
</tbody>
</table>
WHY GRIDS?

- Natural phenomena are often continuous (infinite resolution)
- Computers are obviously restricted to work with a finite amount of data

⇒ Sampling is necessary

CHOOSING THE RIGHT DATA STRUCTURE

- Which data organization is optimal?
  - Is there a neighborhood relationship?
  - How is the neighborhood information stored?
  - How is navigation within the data possible?
  - What calculations are possible with the data?
  - Are the locations of the observation points structured?
**2D Grids**

- Uniform rectilinear
  - Cells: rectangles
  - Layout: uniform
- Curvilinear
  - Cells: different quads
  - Layout: non-uniform, structured
- Unstructured
  - Cells: polygons (mostly triangles)
  - Layout: unstructured

**3D Grids**

- Uniform rectilinear
  - Cells: regular hexahedra
  - Layout: uniform
- Curvilinear
  - Cells: different hexahedra
  - Layout: non-uniform, structured
- Unstructured
  - Cells: different convex polyhedra
  - Layout: unstructured
LOCAL INTERPOLATION

- Interpolation is an approximate reconstruction of the original signal
  - Input: data values given for cell corners
  - Output: data value for an arbitrary point inside a cell

1D - LINEAR INTERPOLATION

- Compute a new data value based on two known values

\[ f(x) = \alpha y_{i+1} + (1 - \alpha)y_i \]
\[ \alpha = \frac{x - x_i}{x_{i+1} - x_i} \]

\[ x = x_i \Rightarrow \alpha = 0 \Rightarrow f(x) = y_i \]
\[ x = x_{i+1} \Rightarrow \alpha = 1 \Rightarrow f(x) = y_{i+1} \]
2D - BILINEAR INTERPOLATION

\[
\begin{align*}
  f_0 &= \alpha f_{10} + (1 - \alpha) f_{00} \\
  f_1 &= \alpha f_{11} + (1 - \alpha) f_{01} \\
  f(x, y) &= \beta f_1 + (1 - \beta) f_0 \\
\end{align*}
\]

\[
\alpha = \frac{x - x_{00}}{x_{10} - x_{00}}, \quad \beta = \frac{y - y_{00}}{y_{01} - y_{00}}
\]

3D - TRILINEAR INTERPOLATION

4 linear interpolations

\[
\begin{align*}
  f_{00} &= \alpha f_{001} + (1 - \alpha) f_{000} \\
  f_{01} &= \alpha f_{011} + (1 - \alpha) f_{010} \\
  f_{10} &= \alpha f_{101} + (1 - \alpha) f_{100} \\
  f_{11} &= \alpha f_{111} + (1 - \alpha) f_{110} \\
\end{align*}
\]

2 bilinear interpolations

\[
\begin{align*}
  f_0 &= \beta f_{01} + (1 - \beta) f_{00} \\
  f_1 &= \beta f_{11} + (1 - \beta) f_{10} \\
\end{align*}
\]

1 trilinear interpolation

\[
  f = \gamma f_1 + (1 - \gamma) f_0
\]
## Algorithm Types

- Algorithm type based on underlying data
  - Scalar algorithms
  - Vector algorithms
  - Tensor algorithms

### Volume Rendering
**OUTLINE**

- Data Set
- 3D Rendering
- Classification

...in real-time on commodity graphics hardware...

---

**VOLUME RENDERING**

*Image order approach:*

*For each pixel {*
  * calculate color of the pixel*
*}*

---

...
GPU-BASED RAYCASTING

- Has several benefits over slicing
  - Easy to implement
  - Flexible (e.g., adaptive sampling, ERT)
  - Correct perspective projection
  - Native 32-bit compositing

BASIC RAY SETUP / TERMINATION

- Two main approaches:
  - Procedural ray/box intersection
    [Röttger et al., 2003], [Green, 2004]
  - Rasterize bounding box
    [Krüger and Westermann, 2003]

- Some possibilities
  - Ray start position and exit check
  - Ray start position and exit position
  - Ray start position and direction vector
**PROCEDURAL RAY SETUP/TERRMINATION**

- 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

---

**"IMAGE-BASED" RAY SETUP/TERRMINATION**

- Rasterize bounding box front faces and back faces [Krüger and Westermann, 2003]
- Ray start position: front faces
- Direction vector: (back faces−front faces)
- Independent of projection (orthogonal/perspective)
**Fragment Shader**

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

```cpp
// In fragment shader code for single-pass ray casting

vec4 main(void) //, float Texcoord ) : TEXCOORD0,
uniform sampler3D SamplerDataVolume,
uniform sampler3D SamplerTransferFunction,
uniform float3 camera,
uniform float4texCoord,
uniform float3 volExtentsMin,
uniform float3 volExtentsMax ) : COLOR

{ float4 value;
  float scalar;
  // Initialize accumulated color and opacity
  float4 color = float4(0, 0, 0, 0);
  // Determine volume entry position
  float3 position = TexCoord.xyz;
  // Compute ray direction
  float3 direction = TexCoord.xyz - camera;
  direction = normalize(direction);
  // Loop for ray traversal
  for (int i = 0; i < 100; i++) // Some large number
  { // Data access to scalar value in 3D volume texture
    value = tex3D(SamplerDataVolume, position);
    scaler = value.a;
    // Apply transfer function
    float3 arc = tex3D(SamplerTransferFunction, color);
    // Front-to-back compositing
    int16_t intArc = int(arc.x);
    int16_t intScal = int(scaler);
    // Advance ray position along ray direction
    position = position + direction * step;
    // Ray termination: Test if outside volume ...
    float3 temp = sign(position - volExtentsMin);
    float3 temp = sign(position - volExtentsMax);
    float3 temp = sign(position - volExtentsMin);
    if (inside > 30.0) // Maximum Intensity Projection
    { return color; }
    // Emission/Absorption
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some color
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some opacity
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some scalar
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some texture
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some transform
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some compositing
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some storage
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some transfer
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some function
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some output
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some calculation
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some code
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some shader
    color = vec4(0.1, 0.1, 0.1, 1.0); // Some fragment
  }
}
```

**Compositing**

- Emission/Absorption
- Maximum Intensity Projection
COLOR MAPPING

- Maps a scalar to a color
- Implemented using lookup table indexed through the scalar value
  \[ \text{index} = n \left( \frac{i - \text{min}}{\text{max} - \text{min}} \right) \]
- Values stored in lookup table
  - RGB or HSV color components
  - Alpha/opacity value
**Mapping to Color**

- To label (indicate set membership)
- To depict a measurement
- To represent reality
- To decorate (consider data/ink ratio)

**Mapping to Color**

- Color perception influenced by fruit picking
- Bad choice: color ramps (rainbow pitfall)
LOOKUP TABLES IN VTK

Example

```cpp
VtkLookupTable* lut = vtkLookupTable::New();
lut->SetHueRange(0.6667, 0.0);
lut->SetSaturationRange(1.0, 1.0);
lut->SetValueRange(1.0, 1.0);
lut->SetNumberOfColors(256);
lut->Build();

vtkPolyDataMapper*
mapper=vtkPolyDataMapper::New();
mapper->SetLookupTable(lut);
mapper->SetScalarRange(min, max);
```

CLASSIFICATION

- Classification is performed through the transfer function
- Intensities are mapped to optical properties
  - Which color to be assigned to which intensities?
  - Which intensities are transparent?
TRANSFER FUNCTIONS IN 2D

TRANSFER FUNCTIONS 3D
VOLUME ILLUMINATION

WHAT IS VOLUMETRIC SHADING?

- Differences to surface shading
  - Light/volume interaction instead of light/surface interaction
  - Translucent materials are frequent
  - Volumetric shadows occur

- Key problems
  - How does light interact with the volume?
  - How can materials be represented?
  - How can scattering and shadowing be supported?
WHY VOLUMETRIC SHADING?

- Langer, Bülthoff: *Depth Discrimination from Shading under Diffuse Lighting* [Perception 00]:
  
  "... depth discrimination under diffuse lighting is superior to that predicted by a classical sunny day model, ..."

- Gribble, Parker: *Enhancing Interactive Particle Visualization with Advanced Shading Models* [APGV06]

EMISSION ABSORPTION MODEL

- Assumption:
  
  Volume consists of small particles which are
  
  - opaque
  - non-reflecting
  - light emitting
  - the only light sources in the scene
**LOCAL VOLUMETRIC SHADING**

- **Illumination limitations** so far
  - Sample points are only light sources
  - No interaction with external light

- Practical solution

\[
I(s_e) = T(s_b, s_e) \cdot I_B + \int_{s_b}^{s_e} T(s', s_e) \cdot \tau(s') \cdot (c_e(s') + c_r(s')) ds'
\]

\[
c_r(s') = \sum_{i=1}^{k} c_{r_i}(s', L_i)
\]

**DIFFUSE LIGHTING**

- Diffuse lighting effects are most prominent in volume data
- Light calculation is based on Lambert’s law

\[
I_d(x) = \left[ L_{d,\text{in}} \cdot k_d \cdot \max\left(\nabla \tau(f(x)) \cdot L, 0\right) \right]
\]
DIFFUSE LIGHTING

no shading  

diffuse lighting dark

/*  
* Returns the diffuse term, considering the  
* currently set OpenGL lighting parameters.  
*  
* @param kd The diffuse color to be used.  
* Usually this is fetched from the transfer  
* function.  
* @param G The computed gradient.  
* @param L The normalized light vector.  
*/  
vec3 getDiffuseColor(in vec3 kd, in vec3 G, in vec3 L) {
  float GdotL = max(dot(G, L), 0.0);
  return kd × lightParams.diffuse.rgb × GdotL;
}

I_d(x) = I_{d,in} \cdot k_d \cdot \max(| \nabla f(x) | \cdot L, 0)
**SPECULAR LIGHTING**

- Specular highlights can add realism to certain tissues
- Lighting calculation is view dependent

\[ I_s(x) = L_{s,in} \cdot k_s \cdot \max\left( \left\vert \nabla \tau(f(x)) \right\vert \cdot H, 0 \right) \]

- Can be expressed as follows

\[ H = \frac{V + L}{2} \]
SPECULAR LIGHTING

```c
/**
 * Returns the specular term, considering the
 * currently set OpenGL lighting parameters.
 *
 * @param ks The specular color to be used.
 * @param G The computed gradient.
 * @param L The normalized light vector.
 * @param V The normalized view vector.
 */
vec3 getSpecularColor(in vec3 ks, in vec3 N, in vec3 L, in vec3 V) {
    vec3 H = normalize(V + L);
    float GdotH = pow(max(dot(G, H), 0.0), matParams.shininess);
    return ks * lightParams.specular.rgb * GdotH;
}

Is(x) = Ls,in \cdot ks \cdot max(\nabla f(x) \cdot H, 0)^{\alpha}
```

AMBIENT LIGHTING

- Add constant light in shadowed regions

```
Ia(x) = L_{a,in} \cdot ka
```

```c
/**
 * Returns the ambient term, considering the
 * currently set OpenGL lighting parameters.
 *
 * @param ka The ambient color to be used.
 * @param fs Usually this is fetched from the transfer
 *       function.
 */
vec3 getAmbientColor(in vec3 ka) {
    return ka * lightParams.ambient.rgb;
}
```

- Drawback: contrast reduction
**AMBIENT LIGHTING**

- ambient dark + diffuse + specular
- ambient medium + diffuse + specular
- ambient bright + diffuse + specular

---

**PHONG LIGHTING**

```cpp
/**
 * Calculates Phong shading.
 *
 * @param G The gradient given in volume object space (does not need to be normalized).
 * @param vpos The voxel position given in volume texture space.
 * @param kd The diffuse material color to be used.
 * @param ks The specular material color to be used.
 * @param ka The ambient material color to be used.
 *
 * vec3 phongShading(in vec3 G, in vec3 vpos, in vec3 kd, in vec3 ks, in vec3 ka) {
 *    vec3 L = normalize(lightPosition - vpos);
 *    vec3 V = normalize(cameraPosition - vpos);
 *    vec3 shadedColor = vec3(0.0);
 *    shadedColor += getDiffuseColor(kd, normalize(G), L);
 *    shadedColor += getSpecularColor(ks, normalize(G), L, V);
 *    shadedColor += getAmbientColor(ka);
 *    return shadedColor;
 *}
 */
```
**Adding Attenuation**

```cpp
shadedColor *= getAttenuation(d);
```

```cpp
/**
 * Returns attenuation based on the currently
 * set OpenGL values. Incorporates constant,
 * linear and quadratic attenuation.
 * 
 * @param d Distance to the light source.
 */
float getAttenuation(in float d)
{
    return 1.0 / (lightParams.constantAttenuation +
                   lightParams.linearAttenuation * d +
                   lightParams.quadraticAttenuation * d * d);
}
```

**Phong Shading + Attenuation**

- no shading
- Phong shading
- Phong shading and attenuation
**Gradient Calculation**

- Surface normal is required for diffuse and specular illumination
- The gradient is a good approximation for a surface normal

**Gradient Estimation**

- The gradient vector is the first-order derivative of the scalar field

\[
\nabla f(x) = \begin{pmatrix}
\frac{\partial f(x)}{\partial x} \\
\frac{\partial f(x)}{\partial y} \\
\frac{\partial f(x)}{\partial z}
\end{pmatrix}
\]

- We can estimate the gradient vector using finite differencing schemes
  - Forward/backward differences
  - Central differences

[Levoy, CG&A 1988]
**Back-/Forward Differences**

- **Forward differences**

\[
f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}
\]

- **Backward differences**

\[
f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h}
\]

---

**Forward Differences**

```c
/**
 * Calculate the gradient based on the A channel using forward differences.
 */
vec3 calcGradient(sampler3D volume, vec3 voxPos, float t, vec3 dir) {
    vec3 gradient;
    float v = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters, voxPos).a);
    float v0 = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters, voxPos + vec3(offset.x, 0.0, 0.0)).a);
    float v1 = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters, voxPos + vec3(0, offset.y, 0)).a);
    float v2 = texture1D(transferFunc_, textureLookup3D(volume, volumeParameters, voxPos + vec3(0, 0, offset.z)).a);
    gradient = vec3(v - v0, v - v1, v - v2);
    return gradient;
}
```
CENTRAL DIFFERENCES

\[ \nabla f(x, y, z) \approx \frac{1}{2h} \begin{pmatrix} f(x + h, y, z) - f(x - h, y, z) \\ f(x, y + h, z) - f(x, y - h, z) \\ f(x, y, z + h) - f(x, y, z - h) \end{pmatrix} \]

WRAP UP
WRAP UP

- Visualization of spatial data
- Scalar visualization algorithms
- GPU-based volume rendering
- Classification
- Volume illumination

LITERATURE

ACKNOWLEDGEMENTS

- These slides contain figures credited to
  - Christof Rezk-Salama
  - Markus Hadwiger
  - Klaus Engel