Read (required):

- Programming Massively Parallel Processors book, Chapter 4 (CUDA Threads)
- Programming Massively Parallel Processors book, Chapter 5 (CUDA Memories)
GPU Texturing

Rage / id Tech 5 (id Software)
Remember: Basic Shading

- Flat shading
  - compute light interaction per polygon
  - the whole polygon has the same color
- Gouraud shading
  - compute light interaction per vertex
  - interpolate the colors
- Phong shading
  - interpolate normals per pixel
- Remember: difference between
  - Phong Lighting Model
  - Phong Shading
Traditional OpenGL Lighting

- Phong lighting model at each vertex (glLight, …)
- Local model only (no shadows, radiosity, …)
- ambient + diffuse + specular (glMaterial!)

Fixed function: Gouraud shading

- Note: need to interpolate specular separately!
- Phong shading: evaluate Phong lighting model in fragment shader (per-fragment evaluation!)
Why Texturing?

- Idea: enhance visual appearance of surfaces by applying fine / high-resolution details
OpenGL Texture Mapping

- Basis for most real-time rendering effects
- Look and feel of a surface
- Definition:
  - A *regularly sampled function* that is mapped onto every *fragment* of a surface
  - Traditionally an image, but…
- Can hold arbitrary information
  - Textures become general data structures
  - Sampled and interpreted by fragment programs
  - Can render into textures → important!
Types of Textures

- **Spatial layout**
  - Cartesian grids: 1D, 2D, 3D, 2D_ARRAY, …
  - Cube maps, …

- **Formats (too many), e.g. OpenGL**
  - GL_LUMINANCE16_ALPHA16
  - GL_RGB8, GL_RGBA8, …: integer texture formats
  - GL_RGB16F, GL_RGBA32F, …: float texture formats
  - compressed formats, high dynamic range formats, …

- **External (CPU) format vs. internal (GPU) format**
  - OpenGL driver converts from external to internal
Texturing: General Approach

Texture space $(u,v)$

Object space $(x_O, y_O, z_O)$

Image Space $(x_I, y_I)$

Parametrization

Rendering (Projection etc.)
Texture Mapping

2D (3D) Texture Space
  | Texture Transformation
2D Object Parameters
  | Parameterization
3D Object Space
  | Model Transformation
3D World Space
  | Viewing Transformation
3D Camera Space
  | Projection
2D Image Space

Kurt Akeley, Pat Hanrahan
2D Texture Mapping

For each fragment:
interpolate the
texture coordinates
(barycentric)

Or:
Use arbitrary, computed coordinates

Texture-Lookup:
interpolate the
texture data
(bi-linear)

Or:
Nearest-neighbor for “array lookup”
3D Texture Mapping

For each fragment:
interpolate the texture coordinates (barycentric)

Or:
Use arbitrary, computed coordinates

Texture-Lookup:
interpolate the texture data (tri-linear)

Or:
Nearest-neighbor for “array lookup”
Where do texture coordinates come from?

- **Online**: texture matrix/texcoord generation
- **Offline**: manually (or by modeling program)

Types:
- spherical
- cylindrical
- planar
- natural
Texture Projectors

Where do texture coordinates come from?

- **Offline**: manual UV coordinates by DCC program
- **Note**: a modeling problem!

![Image of 3D model and UV coordinates grid]
Texture Wrap Mode

- How to extend texture beyond the border?
- Border and repeat/clamp modes
- Arbitrary \((s,t,\ldots) \rightarrow [0,1] \times [0,1] \rightarrow [0,255] \times [0,255]\)

repeat   mirror/repeat  clamp   border

![repeat example](image1)
![mirror/repeat example](image2)
![clamp example](image3)
![border example](image4)
Texture Reconstruction: Magnification

- Bilinear reconstruction for texture magnification ($D < 0$) ("upsampling")
- Weight adjacent texels by distance to pixel position

Texture space

$$T(u+du, v+dv) = du \cdot dv \cdot T(u+1, v+1) + du \cdot (1-dv) \cdot T(u+1, v) + (1-du) \cdot dv \cdot T(u, v+1) + (1-du) \cdot (1-dv) \cdot T(u, v)$$

Vienna University of Technology
Magnification (Bilinear Filtering Example)

Original image

Nearest neighbor  Bilinear filtering
Linear Interpolation / Convex Combination

Special case of linear combination

\[ \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n \]

\[ \alpha_i \geq 0 \]

\[ \alpha_1 + \alpha_2 + \cdots + \alpha_n = 1. \]

The weights \( \alpha_i \) are also the (normalized) barycentric coordinates.

They vary linearly → linear interpolation.
Bi-linear Interpolation

nearest-neighbor

Analog in 3D:
  tri-linear interpolation

bi-linear interpolation
Texture Aliasing: Minification

Problem: One pixel in image space covers many texels
Texture Aliasing: Minification

Caused by *undersampling*: texture information is lost.
Texture Anti-Aliasing: Minification

A good pixel value is the weighted mean of the pixel area projected into texture space.
Texture Anti-Aliasing: MIP Mapping

- MIP Mapping ("Multum In Parvo")
  - Texture size is reduced by factors of 2 (downsampling = "many things in a small place")
  - Simple (4 pixel average) and memory efficient
  - Last image is only ONE texel
Thank you.