CS 380 - GPU and GPGPU Programming
Lecture 16+17: GPU Texturing 1+2

Markus Hadwiger, KAUST
Reading Assignment #10 (until April 23)

Read (required):

- **Brook for GPUs: Stream Computing on Graphics Hardware**
  Ian Buck et al.

Read (optional):

- **The Imagine Stream Processor**
  Ujval Kapasi et al.

- **Merrimac: Supercomputing with Streams**
  Bill Dally et al.
Quiz #2: April 23

Organization

• First 30 min of lecture
• No material (book, notes, ...) allowed

Content of questions

• Lectures (both actual lectures and slides)
• Reading assignments
• Programming assignments (algorithms, methods)
• Solve short practical examples
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.)

Texels
Texture Projectors

Where do texture coordinates come from?

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

- spherical
- cylindrical
- planar
- natural
Texture Projectors

Where do texture coordinates come from?

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

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

repeat  mirror/repeat  clamp  border
Texture Reconstruction: Magnification

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

\[
T(u+du,v+dv) = du \cdot dv \cdot T(u+1,v+1) + du \cdot (1-dv) \cdot T(u+1,v) + (1-du) \cdot dv \cdot T(u,v+1) + (1-du) \cdot (1-dv) \cdot T(u,v)
\]
Magnification (Bilinear Filtering Example)

Original image

Nearest neighbor  Bilinear filtering
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
Texture Anti-Aliasing: MIP Mapping

- MIP Mapping Algorithm
  \[ D := \log(\max(d_1, d_2)) \]
  
- \( T_0 := \) value from texture \( D_0 = \text{trunc} (D) \)

- Use bilinear interpolation

\[ d_1 \]
\[ d_2 \]
Texture Anti-Aliasing: MIP Mapping

- **Trilinear interpolation:**
  - \( T_1 := \text{value from texture} \ D_1 = D_0 + 1 \) (bilin.interpolation)
  - Pixel value := \((D_1 - D) \cdot T_0 + (D - D_0) \cdot T_1\)
  - Linear interpolation between successive MIP Maps
  - Avoids "Mip banding" (but doubles texture lookups)
Texture Anti-Aliasing: MIP Mapping

Other example for bilinear vs. trilinear filtering
Anti-Aliasing: Anisotropic Filtering

- Anisotropic filtering
  - View-dependent filter kernel
  - Implementation: *summed area table*, "RIP Mapping", *footprint assembly*, *elliptical weighted average* (EWA)

Texture space
Anti-Aliasing: Anisotropic Filtering

Example
Texture Anti-aliasing

- Basically, everything done in hardware
- `gluBuild2DMipmaps()` generates MIPmaps
- Set parameters in `glTexParameter()`
  - `GL_LINEAR_MIPMAP_NEAREST`
  - `GL_TEXTURE_MAG_FILTER`
- Anisotropic filtering is an extension:
  - `GL_EXT_texture_filter_anisotropic`
- Number of samples can be varied (4x, 8x, 16x)
  - Vendor specific support and extensions
Texture Coordinates

- Specified manually (`glMultiTexCoord()`)
- Using classical OpenGL texture coordinate generation
  - Linear: from object or eye space vertex coords
  - Special texturing modes (env-maps)
  - Can be further modified with texture matrix
    - E.g., to add texture animation
  - Can use 3rd or 4th texture coordinate for projective texturing!
- Shader allows complex texture lookups!
Thank you.