CS 380 - GPU and GPGPU Programming
Lecture 12: GPU Texturing 2

Markus Hadwiger, KAUST
Reading Assignment #7 (until Mar. 16)

Read (required):

• Interpolation for Polygon Texture Mapping and Shading, Paul Heckbert and Henry Moreton
  http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7886

• MIP-Map Level Selection for Texture Mapping
  http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=765326

Read (optional):

• Frame buffer objects extension specification
  http://www.opengl.org/registry/specs/ARB/framebuffer_object.txt
Magnification (Bilinear Filtering Example)

Original image

Nearest neighbor  Bilinear filtering
Texture Reconstruction: Magnification

- Bilinear reconstruction for texture magnification \((D<0)\) ("upsampling")

- Weight adjacent texels by distance to pixel position

\[
T(u+du,v+dv) = du \cdot dv \cdot T(u+1,v+1) + du \cdot (1-dv) \cdot T(u+1,v) + (1-du) \cdot dv \cdot T(u,v+1) + (1-du) \cdot (1-dv) \cdot T(u,v)
\]
Texture Aliasing: Minification

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

- Caused by *undersampling*: texture information is lost
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

Total size = 4/3
Texture Anti-Aliasing: MIP Mapping

- MIP Mapping Algorithm
- \( D := ld(max(d_1,d_2)) \)
- \( T_0 := \text{value from texture} \quad D_0 = \text{trunc} (D) \)
- Use \textit{bilinear interpolation}
• Use the partial derivatives of texture coordinates with respect to screen space coordinates

$$\begin{vmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y}
\end{vmatrix} = \begin{pmatrix} s_x & s_y \\ t_x & t_y \end{pmatrix}$$

• This is the Jacobian matrix

• Area of parallelogram is the absolute value of the Jacobian determinant (the *Jacobian*)
MIP-Map Level Computation (OpenGL)

- OpenGL 4.5 core specification, pp. 241-243

\[
\lambda_{\text{base}}(x, y) = \log_2[\rho(x, y)]
\]

\[
\rho = \max \left\{ \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2} \right\}
\]

- Does not use area of parallelogram but greater hypotenuse [Heckbert, 1983]

- Approximation without square-roots

\[
m_u = \max \left\{ \left| \frac{\partial u}{\partial x} \right|, \left| \frac{\partial u}{\partial y} \right| \right\} \quad m_v = \max \left\{ \left| \frac{\partial v}{\partial x} \right|, \left| \frac{\partial v}{\partial y} \right| \right\} \quad m_w = \max \left\{ \left| \frac{\partial w}{\partial x} \right|, \left| \frac{\partial w}{\partial y} \right| \right\}
\]

\[
\max\{m_u, m_v, m_w\} \leq f(x, y) \leq m_u + m_v + m_w
\]
• Level of detail value is fractional!
• Use fractional part to blend (lin.) between two adjacent mipmap levels
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)
Anisotropic Filtering: Footprint Assembly

- MIP-map Sample Points
- Line of Anisotropy
- Initial Texture Coordinate

Filter Kernels
- Trilinear
- Anisotropic
Anti-Aliasing: Anisotropic Filtering

Example
Texture Anti-aliasing

- Basically, everything done in hardware
- `gluBuild2DMipmaps()` generates MIPmaps
- Set parameters in `glTexParameteri()`
  - `GL_TEXTURE_MAG_FILTER`: GL_NEAREST, GL_LINEAR, ...
  - `GL_TEXTURE_MIN_FILTER`: GL_LINEAR_MIPMAP_NEAREST
- 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.