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

Markus Hadwiger, KAUST
Reading Assignment #10 (until Nov 4)

Read (required):

- **Brook for GPUs: Stream Computing on Graphics Hardware**
  Ian Buck et al., SIGGRAPH 2004
  http://graphics.stanford.edu/papers/brookgpu/

Read (optional):

- **The Imagine Stream Processor**
  Ujval Kapasi et al.; IEEE ICCD 2002

- **Merrimac: Supercomputing with Streams**
  Bill Dally et al.; SC 2003
  https://dl.acm.org/citation.cfm?doid=1048935.1050187
Programming Assignments: Schedule (tentative)

Assignment #1:
• Querying the GPU (OpenGL/GLSL and CUDA) due Sep 4

Assignment #2:
• Phong shading and procedural texturing (GLSL) due Sep 24

Assignment #3:
• Image Processing with GLSL due Oct 7

Assignment #4:
• Image Processing with CUDA
• Convolutional layers with CUDA due Oct 21

Assignment #5:
• Linear Algebra (CUDA) due Nov 11
GPU Texturing

Rage / id Tech 5 (id Software)
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 \((s_0, t_0)\) to \((s, t)\) to \((s_1, t_1)\) (barycentric)
- Use arbitrary, computed coordinates

**Texture-Lookup:**
- Interpolate the texture data \((s, t)\) (bi-linear)
- Nearest-neighbor for “array lookup”
Linear Interpolation / Convex Combinations

Linear interpolation in 1D:

\[ f(\alpha) = (1 - \alpha)v_1 + \alpha v_2 \]

Line embedded in 2D (linear interpolation of vertex coordinates/attributes):

\[ f(\alpha_1, \alpha_2) = \alpha_1 v_1 + \alpha_2 v_2 \]

\[ \alpha_1 + \alpha_2 = 1 \]

\[ f(\alpha) = v_1 + \alpha (v_2 - v_1) \]

\[ \alpha = \alpha_2 \]

Line segment: \( \alpha_1, \alpha_2 \geq 0 \) (→ convex combination)

Compare to line parameterization with parameter \( t \):

\[ v(t) = v_1 + t(v_2 - v_1) \]
**Linear** combination \((n\text{-dim. space})\):

\[ \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i \]

**Affine** combination: Restrict to \((n-1)\text{-dim. subspace})\):

\[ \alpha_1 + \alpha_2 + \ldots + \alpha_n = \sum_{i=1}^{n} \alpha_i = 1 \]

**Convex** combination:

\[ \alpha_i \geq 0 \]

(restrict to simplex in subspace)
Linear Interpolation / Convex Combinations

\[ \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i \]

\[ \alpha_1 + \alpha_2 + \ldots + \alpha_n = \sum_{i=1}^{n} \alpha_i = 1 \]

Re-parameterize to get affine coordinates:

\[ \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = \]
\[ \tilde{\alpha}_1 (v_2 - v_1) + \tilde{\alpha}_2 (v_3 - v_1) + v_1 \]
\[ \tilde{\alpha}_1 = \alpha_2 \]
\[ \tilde{\alpha}_2 = \alpha_3 \]
The weights $\alpha_i$ are the (normalized) barycentric coordinates
\[ \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i \]
\[ \alpha_1 + \alpha_2 + \ldots + \alpha_n = \sum_{i=1}^{n} \alpha_i = 1 \]
\[ \alpha_i \geq 0 \]
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
Linear Perspective

Correct Linear Perspective

Incorrect Perspective

Linear Interpolation, Bad
Perspective Interpolation, Good

Kurt Akeley, Pat Hanrahan
Texture Mapping Polygons

Forward transformation: linear projective map

$$\begin{bmatrix}
  x \\
  y \\
  w \\
\end{bmatrix} = \begin{bmatrix}
  a & b & c & s \\
  d & e & f & t \\
  g & h & i & r \\
\end{bmatrix}$$

Backward transformation: linear projective map

$$\begin{bmatrix}
  s \\
  t \\
  r \\
\end{bmatrix} = \begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i \\
\end{bmatrix}^{-1} \begin{bmatrix}
  x \\
  y \\
  w \\
\end{bmatrix}$$

Kurt Akeley, Pat Hanrahan
Incorrect attribute interpolation

$A' \neq A$ !

Kurt Akeley, Pat Hanrahan
Linear interpolation

Compute intermediate attribute value

- Along a line: \( A = aA_1 + bA_2, \quad a+b=1 \)
- On a plane: \( A = aA_1 + bA_2 + cA_3, \quad a+b+c=1 \)

Only projected values interpolate linearly in screen space (straight lines project to straight lines)

- \( x \) and \( y \) are projected (divided by \( w \))
- Attribute values are not naturally projected

Choice for attribute interpolation in screen space

- Interpolate unprojected values
  - Cheap and easy to do, but gives wrong values
  - Sometimes OK for color, but
  - Never acceptable for texture coordinates

- Do it right

Kurt Akeley, Pat Hanrahan
Linear Perspective

Correct Linear Perspective

Incorrect Perspective

Linear Interpolation, Bad
Perspective Interpolation, Good

Kurt Akeley, Pat Hanrahan
Perspective Texture Mapping

Linear interpolation in object space

\[
\frac{ax_1 + bx_2}{aw_1 + bw_2} \neq a \frac{x_1}{w_1} + b \frac{x_2}{w_2}
\]

Linear interpolation in screen space

a = b = 0.5
Ultima Underworld (Looking Glass, 1992)
Early Perspective Texture Mapping in Games

DOOM (id Software, 1993)
Early Perspective Texture Mapping in Games

Quake (id Software, 1996)
Perspective-correct linear interpolation

Only projected values interpolate correctly, so project $A$

- Linearly interpolate $A_1/w_1$ and $A_2/w_2$

Also interpolate $1/w_1$ and $1/w_2$

- These also interpolate linearly in screen space

Divide interpolants at each sample point to recover $A$

- $(A/w) / (1/w) = A$

- Division is expensive (more than add or multiply), so
  - Recover $w$ for the sample point (reciprocate), and
  - Multiply each projected attribute by $w$

Barycentric triangle parameterization:

$$ A = \frac{aA_1/w_1 + bA_2/w_2 + cA_3/w_3}{a/w_1 + b/w_2 + c/w_3} \quad \text{with} \quad a + b + c = 1 $$

Kurt Akeley, Pat Hanrahan
Perspective Texture Mapping

- Solution: interpolate \((s/w, t/w, 1/w)\)
- \((s/w) / (1/w) = s\) etc. at every fragment

Heckbert and Moreton
Perspective-Correct Interpolation Recipe

Heckbert and Moreton

\[
 r_i(x, y) = \frac{r_i(x, y)/w(x, y)}{1/w(x, y)}
\]

1. Associate a record containing the \( n \) parameters of interest \( (r_1, r_2, \ldots, r_n) \) with each vertex of the polygon.

2. For each vertex, transform object space coordinates to homogeneous screen space using \( 4 \times 4 \) object to screen matrix, yielding the values \( (xw, yw, zw, w) \).

3. Clip the polygon against plane equations for each of the six sides of the viewing frustum, linearly interpolating all the parameters when new vertices are created.

4. At each vertex, divide the homogeneous screen coordinates, the parameters \( r_i \), and the number 1 by \( w \) to construct the variable list \( (x, y, z, s_1, s_2, \ldots, s_{n+1}) \), where \( s_i = r_i/w \) for \( i \leq n \), \( s_{n+1} = 1/w \).

5. Scan convert in screen space by linear interpolation of all parameters, at each pixel computing \( r_i = s_i/s_{n+1} \) for each of the \( n \) parameters; use these values for shading.
Magnification (Bi-linear Filtering Example)

Original image

Nearest neighbor

Bi-linear filtering
Texture Aliasing: Minification

Problem: One pixel in image space covers many texels
Thank you.