Textures

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html
Texture Mapping

- A technique for specifying variations in surface reflectance properties of an object
- Store the reflectance as an image and “map” it onto the object
- The stored image is called a texture map
Texture Correspondence

- A texture map is defined in its own 2D coordinate system, parameterized by \((u, v)\).
- Establish a correspondence by assigning \((u, v)\) coordinates to triangle vertices.
OpenGL Texturing Snippet

```c
// Enable texture mode.
GLenum textureMode = GL_TEXTURE_2D;

// Create texture object.
GLuint textureID = 0;

// Create pixel data and fill it out.
unsigned char *pixels = new unsigned char[width * height * 4];

// Load pixel data into OpenGL/GPU, 4 bytes/pixel, RGBA.
GLint textureTarget = GL_TEXTURE_2D;

// Activate and bind texture to the multi-texture target 0.
GLint textureUnit = 0;

// Draw the triangle.

// Destory texture once you are completely done with it.
```
Then, for each pixel inside a triangle, calculate the pixel’s \((u,v)\) texture coordinates using barycentric interpolation of the triangle vertices’ texture coordinates.

\[
[u, v] = a_0 [u_0, v_0] + a_1 [u_1, v_1] + a_2 [u_2, v_2]
\]
Pixel Color

- Given the pixel’s \((u, v)\) texture coordinates, use interpolation in the texture map to find the pixel’s color.

![Image of texture map with pixel location indicated]
Pixel Color: Nearest Neighbor

GL_NEAREST

GL_LINEAR
Pixel Color: Bilinear Interpolation

Linearly Interpolate
Pixel Color: Bilinear Interpolation
Nearest Neighbor Vs Bilinear

GL_NEAREST

GL_LINEAR
Nearest Neighbor Vs Bilinear

More on this when we discuss “Sampling”
Screen Space vs. World Space

- Triangles change shape nonlinearly via perspective transformation, leading to different barycentric weights before and after the perspective transformation.
- Interpolating in screen space results in texture distortion.
- Interpolating in world space requires projecting all pixel locations backwards from screen space to world space, which is expensive.
Texture Distortion

\[ t: \text{Screen-space parameter} \]

\[ s: \text{World-space parameter} \]
Screen-space and World-space parameters don’t match!
Texture Distortion

- Uniform increments along the edge in world space do not correspond to uniform increments along the edge in screen space.
- Barycentric interpolation (which is linear) does not account for this nonlinearity.
Mesh Refinement

- Refinement of the triangle mesh improves the result.
- A nonlinear function can be approximated as a piecewise linear function if the intervals are small enough.
- However, some errors are still obvious, especially at T-junctions where levels-of-refinement change.

![Diagram showing mesh refinement process](image)
Mesh Refinement

Does not work!
Perspective Correct Interpolation

- Find the relationship between the barycentric weights in screen space and those in world space
- Use this relationship to compute the world space barycentric weights from the screen space barycentric weights
Perspective Correct Interpolation

Two points in world space

\[ p_1^w = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} \quad p_2^w = \begin{bmatrix} x_2 \\ z_2 \end{bmatrix} \]
Perspective Correct Interpolation

Two points in world space

\[ p^w_1 = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} \quad p^w_2 = \begin{bmatrix} x_2 \\ z_2 \end{bmatrix} \]

Interpolation in world space

\[ p^w(t) = (1 - t) \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + t \begin{bmatrix} x_2 \\ z_2 \end{bmatrix} \]
Perspective Correct Interpolation

Two points in world space

\[
\begin{align*}
p_1^w &= \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} & p_2^w &= \begin{bmatrix} x_2 \\ z_2 \end{bmatrix}
\end{align*}
\]

Interpolation in world space

\[
p_w^w(t) = (1 - t) \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + t \begin{bmatrix} x_2 \\ z_2 \end{bmatrix}
\]

Project the interpolated point on the screen

\[
\text{Proj}(p_x^w(t)) = d \frac{(1 - t)x_1 + tx_2}{(1 - t)z_1 + tz_2}
\]
Perspective Correct Interpolation

Two points in world space

\[ p_1^w = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} \quad p_2^w = \begin{bmatrix} x_2 \\ z_2 \end{bmatrix} \]

Interpolation in world space

\[ p^w(t) = (1 - t) \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + t \begin{bmatrix} x_2 \\ z_2 \end{bmatrix} \]

Project the interpolated point on the screen

\[ \text{Proj}(p^w_x(t)) = d \frac{(1 - t)x_1 + tx_2}{(1 - t)z_1 + tz_2} \]
Perspective Correct Interpolation

Interpolation of the same two points in screen space (after projection)

\[ P_x^s(s) = (1 - s) \frac{dx_1}{z_1} + s \frac{dx_2}{z_2} \]
Perspective Correct Interpolation

Interpolation of the same two points in screen space (after projection)

\[ P^s_x(s) = (1 - s) \frac{dx_1}{z_1} + s \frac{dx_2}{z_2} \]

Screen space point and world-space point after projection must match

\[ d \frac{(1 - t)x_1 + tx_2}{(1 - t)z_1 + tz_2} = (1 - s) \frac{dx_1}{z_1} + s \frac{dx_2}{z_2} \]
Perspective Correct Interpolation

Interpolation of the same two points in screen space (after projection)

\[ P_s^x(s) = (1 - s) \frac{d x_1}{z_1} + s \frac{d x_2}{z_2} \]

Screen space point and world-space point after projection must match

\[ d \frac{(1 - t)x_1 + tx_2}{(1 - t)z_1 + tz_2} = (1 - s) \frac{dx_1}{z_1} + s \frac{dx_2}{z_2} \]

After algebra

\[ t = \frac{s z_1}{z_2 + s(z_1 - z_2)} \]
Perspective Correct Interpolation

Screen Space Triangle

\[(x'_0, z'_0)\]

\[(x'_1, z'_1)\]

\[(x'_2, z'_2)\]

\[a_0^w = \frac{z_1 z_2 a_0}{a_0 z_1 z_2 + a_1 z_0 z_2 + (1 - a_0 - a_1) z_0 z_1}\]

\[a_1^w = \frac{z_0 z_2 a_1}{a_1 z_0 z_2 + a_0 z_1 z_2 + (1 - a_0 - a_1) z_0 z_1}\]

World Space Triangle

\[(x_0, z_0)\]

\[(x_1, z_1)\]

\[(x_2, z_2)\]
Perspective Correct Interpolation

Finally!

\[ [u, v] = a_0^w [u_0, v_0] + a_1^w [u_1, v_1] + a_2^w [u_2, v_2] \]
Perspective Correct Interpolation

Finally!

$$[u, v] = a_0^w [u_0, v_0] + a_1^w [u_1, v_1] + a_2^w [u_2, v_2]$$
Aliasing

What we get

What we want
Aliasing

Small image

Large texture

Source pixels covers many destination pixels
Aliasing

Small image  Large texture

Source pixels covers many destination pixels

More when we discuss sampling
MIP Maps

- Multum in Parvo: Much in little, many in small places
- Precomputes the texture maps at multiple resolutions, using averaging as a low pass filter
- When texture mapping, choose the image size that approximately gives a 1 to 1 pixel to texel correspondence
- The averaging “bakes-in” all the nearby pixels that otherwise would not be sampled correctly
MIP Maps

- 4 neighboring pixels of the higher level are averaged to form a single pixel in the lower level
- Starting at a base resolution, you can store EVERY coarser resolution in powers of 2
MIP Maps

- 4 neighboring pixels of the higher level are averaged to form a single pixel in the lower level
- Starting at a base resolution, you can store EVERY coarser resolution in powers of 2

Only 33% memory overhead
MIP Maps

point sampling

using mip-maps
MIP Maps

point sampling

using mip-maps
Assigning Texture Coordinates

- For certain surfaces, the \((u, v)\) texture coordinates can be generated procedurally.
- Example: Cylinder
  - Map the \(u\) coordinate from \([0, 1]\) to \([0, 2\pi]\) for \(\Phi\).
  - Map the \(v\) coordinate from \([0, 1]\) to \([0, h]\) for \(y\).
  - This wraps the image around the cylinder.
- For more complex surfaces, \((u, v)\) must be defined per vertex manually or by using proxy objects.
Proxy Objects – Step 1

- Assign texture coordinates to intermediate/proxy objects:
  - Example: Cylinder
    - wrap texture around the outside of the cylinder
    - not the top or bottom, in order to avoid distorting the texture
  - Example: Cube
    - unwrap the cube and map texture over the unwrapped cube
    - the texture is seamless across some of the edges, but not necessarily others
Proxy Objects – Step 2

- Map texture coordinates from the intermediate/proxy object to the final object
- Three ways of mapping are typically used
  - Use the intermediate/proxy object’s surface normal
  - Use the target object’s surface normal
  - Use rays emanating from center of target object
Cube Mapping

```
glTexGenfv(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
glTexGenfv(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
glTexGenfv(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
```

http://learnopengl.com/#!Advanced-OpenGL/Cubemaps
Projective Texturing

- Treat light Source as a
- Render the scene normally from the actual camera

Segal et. al. SIGGRAPH’92
Projective Texturing

• Assign Texture Coordinates \((s,t,r)\) to position \((x,y,z)\)

```c
float [] planeS = { 1.0f, 0.0f, 0.0f, 0.0f };
glTexGenfv(GL_S, GL_OBJECT_PLANE, planeS);

float [] planeT = { 0.0f, 1.0f, 0.0f, 0.0f };
glTexGenfv(GL_T, GL_OBJECT_PLANE, planeT);

float [] planeR = { 0.0f, 0.0f, 1.0f, 0.0f };
glTexGenfv(GL_R, GL_OBJECT_PLANE, planeR);

glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);

\[
\begin{bmatrix}
  s \\
  t \\
  r
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\]

ObjectSpace
```
Projective Texturing

• Assign Texture Coordinates \((s, t, r)\) to position \((x, y, z)\)

\[
\begin{bmatrix}
  s \\
  t \\
  r
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} \text{ ObjectSpace}
\]
Projective Texturing

• Assign Texture Coordinates \((s,t,r)\) to position \((x,y,z)\)

\[
\begin{bmatrix}
s \\ t \\ r \\
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\
\end{bmatrix} \begin{bmatrix}
x \\ y \\ z \\
\end{bmatrix}
\]

ObjectSpace
Projective Texturing

- Assign Texture Coordinates \((s, t, r)\) to position \((x, y, z)\)

```
// OpenGL command to enable texture generation
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);

// Example code to assign texture coordinates
float [] planeS = { 1.0f, 0.0f, 0.0f, 0.0f };
glTexGenfv(GL_S, GL_OBJECT_PLANE, planeS);
```

Mathematically:

\[
\begin{bmatrix}
  s \\
  t \\
  r
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\]

So much work just to say \((s, t, r) = (x, y, z)\)

Much easily done in newer OpenGL
Projective Texturing

- Use a similar View (from the light’s point of view) and Projection matrix to transform texture coordinates to NDC (-1, 1)

\[
\begin{bmatrix}
  s' \\
  t' \\
  r' \\
  q'
\end{bmatrix}_{NDC} = P \cdot V \cdot M \begin{bmatrix}
  s \\
  t \\
  r \\
  1
\end{bmatrix}
\]
Projective Texturing

• Map NDC (-1, 1) to Texture Coordinate space (0-1)
  • Scale and add Bias

\[
\begin{bmatrix}
  s'' \\
  t'' \\
  r'' \\
  q'' \end{bmatrix}_{\text{TextureSpace}} =
\begin{bmatrix}
  0.5 & 0 & 0 & 0.5 \\
  0 & 0.5 & 0 & 0.5 \\
  0 & 0 & 0.5 & 0.5 \\
  0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix}
  s' \\
  t' \\
  r' \\
  q' \end{bmatrix}_{\text{NDC}}
\]
Projective Texturing

• Map NDC \((-1, 1)\) to Texture Coordinate space \((0-1)\)
  
  • Scale and add Bias

\[
\begin{bmatrix}
  s'' \\
  t'' \\
  r'' \\
  q''
\end{bmatrix}_{\text{TextureSpace}} =
\begin{bmatrix}
  0.5 & 0 & 0 & 0.5 \\
  0 & 0.5 & 0 & 0.5 \\
  0 & 0 & 0.5 & 0.5 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  s' \\
  t' \\
  r' \\
  q'
\end{bmatrix}_{\text{NDC}}
\]

Final texture coordinates after perspective-correct interpolation of \((s'', t'', r'', q'')\)

\[
\begin{pmatrix}
  s'' \\
  t'' \\
  r'' \\
  q''
\end{pmatrix}
\begin{pmatrix}
  s'' \\
  t'' \\
  r'' \\
  q''
\end{pmatrix}
\]
Projective Texturing

• Map NDC (-1, 1) to Texture Coordinate space (0-1)
  • Scale and add Bias

\[
\begin{bmatrix}
s'' \\
q''
\end{bmatrix}
\text{TextureSpace}
\begin{bmatrix}
0 & 0 & 0.5 & 0.5 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
s' \\
q'
\end{bmatrix}
\text{NDC}
\]

**Hardware does it for you :)**

Final texture coordinates after perspective-correct interpolation of \((s'', t'', r'', q'')\)

\[
\begin{pmatrix}
s'' & t'' & r'' \\
q'' & q'' & q''
\end{pmatrix}
\]
How To Set Texture Matrices?

glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glTranslatef(0.5, 0.5, 0.5);
glScale3f(0.5, 0.5, 0.5);
gluPerspective(...);
gluLookAt(...);
glLoadMatrixf(modelMatrix);
.
.
glMatrixMode(GL_MODELVIEW);
How To Set Texture Matrices?

```
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glTranslatef(0.5, 0.5, 0.5);
if (false) {
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
}
```

Much simpler in newer OpenGL (will discuss later)
Projective Texturing: Issues

\[ q'' < 0 \]
Projective Texturing: Issues

Severe Aliasing
Textures

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html