Texture Mapping

- Associate 2D information with 3D surface
  - Point on surface corresponds to a point in the texture
- Introduced to increase realism
  - Lighting/shading models not enough
- Hide geometric simplicity
  - Map a brick wall texture on a flat polygon
  - Create *bumpy* effect on surface

Texture Pipeline

1. Compute object space location
2. Use projector function to find \((u, v)\)
3. Use corresponder function to find texels
4. Apply value transform function (e.g., scale, bias)
5. Modify illumination equation value

Texture Mapping

- \((s, t)\) parameterization in OpenGL

Object position: \((-2.3, 7.1, 17.7)\)
Parameter space: \((0.32, 0.29)\)
Image space: \((81, 74)\)
Texel color: \((0.9, 0.8, 0.7)\)
Texture Mapping

- **Texture Coordinates**
  - generation at vertices
    - specified by programmer or artist
    - \( \text{glTexCoord2f}(s, t) \)
    - \( \text{glVertexf}(x, y, z) \)
  - generate as a function of vertex coords
    - \( \text{glTexGeni}(), \text{glTexGenf}() \)
    - \( s = ax + by + cz + dh \)
  - interpolated across triangle (like R,G,B,Z)
    - ...well not quite!

Texture Coordinate Interpolation

- Perspective Correct Interpolation
  - \( \alpha, \beta, \gamma \):
    - Barycentric coordinates of a point \( P \) in a triangle
  - \( s0, s1, s2 \): texture coordinates
  - \( w0, w1, w2 \): homogeneous coordinates

\[
 s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}
\]

Attribute Interpolation

- Bilinear Interpolation
  - Incorrect
  - Perspective correct
  - Correct

Texture Coordinate Transformation

- **Textures of other dimensions**
  - 3D: solid textures
    - e.g.: wood grain, medical data, ...
    - \( \text{glTexCoord3f}(s, t, r) \)
  - 4D: 3D + time, projecting textures
    - \( \text{glTexCoord3f}(s, t, r, q) \)

- **Motivation**:
  - Change scale, orientation of texture on an object

- **Approach**:
  - \( \text{texture matrix stack} \)
  - \( 4x4 \) matrix stack
  - transforms specified (or generated) tex coords
    - \( \text{glMatrixMode} (\text{GL_TEXTURE}) ; \)
    - \( \text{glLoadIdentity} () ; \)
    - ...
Texture Coordinate Transformation

- Example:

\[
\begin{align*}
(0,0) & \rightarrow (0,0) \\
(0,1) & \rightarrow (0,1) \\
(1,0) & \rightarrow (4,0) \\
(1,1) & \rightarrow (4,4)
\end{align*}
\]

\[\text{glScalef}(4.0, 4.0, ?);\]

Texture Lookup

- Issue:
  - What happens to fragments with \( s \) or \( t \) outside the interval \([0...1]\)?

Multiple choices:
- Take only fractional part of texture coordinates
  - Cyclic repetition of texture to tile whole surface
    \[\text{glTexParameteri( \ldots, \text{GL_TEXTURE_WRAP_S}, \text{GL_REPEAT})}\]
  - Clamp every component to range \([0...1]\)
    \[\text{glTexParameteri( \ldots, \text{GL_TEXTURE_WRAP_S}, \text{GL_CLAMP})}\]

Texture Functions

- Once got value from the texture map, can:
  - Directly use as surface color \text{GL_REPLACE}
  - Modulate surface color \text{GL_MODULATE}
  - Blend surface and texture colors \text{GL_DECAL}
  - Blend surface color with another \text{GL_BLEND}

- Specific action depends on internal texture format
  - Only existing channels used

- Specify with \text{glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode)}

Reconstruction

- How to deal with:
  - pixels that are much larger than texels?
    - apply filtering, “averaging”
  - pixels that are much smaller than texels?
    - interpolate

Mip-mapping

Use an “image pyramid” to pre-compute averaged versions of the texture

Problem:
- A MIP-map level selects the same minification factor for both the \( s \) and the \( t \) direction (isotropic filtering)
- In reality, perspective foreshortening (amongst other reasons) can cause different scaling factors for the two directions
Mip-mapping

- Which resolution to choose:
  - MIP-mapping: take resolution corresponding to the smaller of the sampling rates for \( s \) and \( t \)
    - Avoids aliasing in one direction at cost of blurring in the other direction
  - Better: anisotropic texture filtering
    - Also uses MIP-map hierarchy
    - Choose larger of sampling rates to select MIP-map level
    - Then use more samples for that level to avoid aliasing
    - Maximum anisotropy (ratio between \( s \) and \( t \) sampling rate) usually limited (e.g. 4 or 8)

Texture Mapping Functions

Two Step Parameterization:
- Step 1: map 2D texture onto an intermediate simple surface
  - Sphere
  - Cube
  - Cylinder
- Step 2: map from this surface to the object
  - Surface normal
- Commonly used for environment mapping

Environment Mapping

- Two Step Parameterization:
  - Step 1: map 2D texture onto an intermediate simple surface
    - Sphere
    - Cube
    - Cylinder
  - Step 2: map from this surface to the object
    - Surface normal
- Commonly used for environment mapping

Spherical Maps – Blinn & Newell ’76

- Transform reflection vector \( r \) into spherical coordinates \((\theta, \phi)\)
  - \( \theta \) varies from \([0, \pi]\) (latitude)
  - \( \phi \) varies from \([0, 2\pi]\) (longitude)
  
  \[
  r = (r_x, r_y, r_z) = 2(n \cdot v)n - v
  \]

  \[
  \Theta = \arccos(-r_z)
  \]

  \[
  \Phi = \begin{cases} 
  \arccos(-r_x / \sin \Theta) & \text{if } r_y \geq 0 \\
  2\pi - \arccos(-r_x / \sin \Theta) & \text{otherwise}
  \end{cases}
  \]

Spherical Maps – Blinn & Newell ’76

- Slice through the photo
  - Each pixel corresponds to particular direction in the environment
- Singularity at the poles!
- OpenGL support \texttt{GL_SPHERE_MAP}

Cube Mapping – Greene ’86

- A slice through the photo
  - Each pixel corresponds to a particular direction in the environment

- \texttt{Singularity} at the poles!
- OpenGL support \texttt{GL_SPHERE_MAP}
Cube Mapping – Greene ‘86

- Direction of reflection vector $r$ selects the face of the cube to be indexed
  - Co-ordinate with largest magnitude
    - e.g., the vector (-0.2, 0.5, -0.84) selects the –Z face!
  - Remaining two coordinates (normalized by the 3rd coordinate) selects the pixel from the face.
    - e.g., (-0.2, 0.5) gets mapped to (0.38, 0.80).

- Difficulty in interpolating across faces!
- OpenGL support GL_CUBE_MAP