Texture Mapping (cont.)

Jian Huang
CS 456
OpenGL functions

• During initialization read in or create the texture image and place it into the OpenGL state.

  \[
  \text{glTexImage2D (GL\_TEXTURE\_2D, 0, GL\_RGB,}
  \text{ imageWidth, imageHeight, 0, GL\_RGB,}
  \text{ GL\_UNSIGNED\_BYTE, imageData)};
  \]

• Before rendering your textured object, enable texture mapping and tell the system to use this particular texture.

  \[
  \text{glBindTexture (GL\_TEXTURE\_2D, 13)};
  \]
OpenGL functions

• *During rendering, give the cartesian coordinates and the texture coordinates for each vertex.*

```
    glBegin (GL_QUADS);
    glTexCoord2f (0.0, 0.0);
    glVertex3f (0.0, 0.0, 0.0);
    glTexCoord2f (1.0, 0.0);
    glVertex3f (10.0, 0.0, 0.0);
    glTexCoord2f (1.0, 1.0);
    glVertex3f (10.0, 10.0, 0.0);
    glTexCoord2f (0.0, 1.0);
    glVertex3f (0.0, 10.0, 0.0);
    glEnd ();
```
OpenGL functions

• Automatic texture coordinate generation
  – Void glTexGenf( coord, pname, param)
    • Coord:
      – GL_S, GL_T, GL_R, GL_Q
    • Pname
      – GL_TEXTURE_GEN_MODE
      – GL_OBJECT_PLANE
      – GL_EYE_PLANE
    • Param
      – GL_OBJECT_LINEAR
      – GL_EYE_LINEAR
      – GL_SPHERE_MAP
What happens when outside the 0-1 range?

• (u,v) should be in the range of 0~1
• What happens when you request (1.5 2.3)?
  – Tile: repeat (OGL); the integer part of the value is dropped, and the image repeats itself across the surface
  – Mirror: the image repeats itself but is mirrored (flipped) on every other repetition
  – Clamp to edge – value outside of the range are clamped to this range
  – Clamp to border – all those outside are rendered with a separately defined color of the border
Methods for modifying surface

- After a texture value is retrieved (may be further transformed), the resulting values are used to modify one or more surface attributes
- Called *combine functions* or *texture blending operations*
  - Replace: replace surface color with texture color
  - Decal: replace surface color with texture color, blend the color with underlying color with an alpha texture value, but the alpha component in the framebuffer is not modified
  - Modulate: multiply the surface color by the texture color (shaded + textured surface)
Multi-Pass Rendering

• The final color of a pixel in the framebuffer is dependent on:
  – The shading/illumination applied on it
  – How those fragments are combined/blended together

• Given the set graphics hardware, how can we get more control (programmability)?
  – Example: color plate V in the book

• Want a range of special effects to be available but tailor based on the variety of hardware?
Multi-pass Rendering

• Each pass renders every pixel once

• Each pass computes a piece of the lighting equation and the framebuffer is used to store intermediate results
Quake III Engine

• Ten passes
  – (1-4: accumulate bump map)
  – 5: diffuse lighting)
  – 6: base texture (with specular component)
  – (7: specular lighting)
  – (8: emissive lighting)
  – (9: volumetric/atmospheric effects)
  – (10: screen flashes)
AB+CD?

• Using framebuffer as the intermediate storage, can multi-pass rendering implement AB+CD?
  – Suppose A, B, C, D are each result of a certain pass
Multi-texture Environments

- Chain of Texture Units (Texture Stages)

Pre-texturing color
\texttt{glColor3f(r,g,b)}

\texttt{glMultiTexCoord2f(GL\_TEXTURE0, ...)}
\texttt{GL\_MODULATE}
\texttt{glMultiTexCoord2f(GL\_TEXTURE1, ...)}
\texttt{GL\_DECAL}
\texttt{glMultiTexCoord2f(GL\_TEXTURE2, ...)}
\texttt{GL\_BLEND}

Post-texturing color
Multi-Texturing

• Should have at least 2 texture units if multi-texturing is supported

• Each texture unit:
  – Texture image
  – Filtering parameters
  – Environment application
  – Texture matrix stack
  – Automatic texture-coordinate generation (doesn’t seem obvious, but very have very creative apps)
Multi-Texturing

• Apply more than one texture to each fragment
  – Alpha
  – Luminance
  – Luminance and alpha
  – Intensity
  – RGB
  – RGBA

• For instance, first put a color texture on a fragment and then put an intensity map on the fragment that modulates the intensity to simulate lighting situation
  – (what could this be?)
Putting Things Together

• With multi-pass rendering + multi-texturing, the lighting equation can be very flexibly controlled to render effects that graphics hardware does not do by its default

• Common practice in the industry now

• Next, let’s look at a few established texturing methods
Alpha Mapping

• Given a square texture, what if we only want the interesting part of it?
  – Say, a tree?

• Utilize the alpha channel of the texture, so that only the interesting part becomes non-transparent

• But what about the order?
Alpha Mapping

• A binary mask, really redefines the geometry.
Light Mapping

• Given a complicated lighting condition, how to accelerate and still get real-time frame rates?

• Assuming your environment is diffuse, let’s compute your lighting condition once and then use as texture map
Example
Choosing a Mapping

• Problem: In a preprocessing phase, points on polygons must be associated with points in maps
• One solution:
  – Find groups of polygons that are “near” co-planar and do not overlap when projected onto a plane
    • Result is a mapping from polygons to planes
  – Combine sections of the chosen planes into larger maps
  – Store texture coordinates at polygon vertices
• Lighting tends to change quite slowly (except at hard shadows), so the map resolution can be poor
Generating the Map

• Problem: What value should go in each pixel of the light map?
• Solution:
  – Map texture pixels back into world space (using the inverse of the texture mapping)
  – Take the illumination of the polygon and put it in the pixel
• Advantages of this approach:
  – Choosing “good” planes means that texture pixels map to roughly square pieces of polygon - good sampling
  – Not too many maps are required, and not much memory is wasted
Example
Example
Applying Light Maps

• Use multi-texturing hardware
  – First stage: Apply color texture map
  – Second stage: Modulate with light map

• Pre-lighting textures:
  – Apply the light map to the texture maps as a pre-process
  – When is this less appealing?

• Multi-pass rendering:
  – Same effect as multi-texturing, but modulating in the frame buffer
Gloss Mapping

• To rendering a tile floor that is worn in places or a sheet of metal with some rusty spots
• Let’s control the specular component of the lighting equation
Bump Mapping

• Many textures are the result of small perturbations in the surface geometry
• Modeling these changes would result in an explosion in the number of geometric primitives.
• Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.
Bump Mapping

- This modifies the surface normals.
- More on this later.
Bump Mapping
Bump Mapping
Bump Mapping

• Consider the lighting for a modeled surface.
Bump Mapping

• We can model this as deviations from some base surface.
• The question is then how these deviations change the lighting.
Bump Mapping

• Assumption: small deviations in the normal direction to the surface.

\[ \bar{\mathbf{X}} = \bar{\mathbf{X}} + B \mathbf{N} \]

Where B is defined as a 2D function parameterized over the surface:

\[ B = f(u,v) \]
Bump Mapping

• Step 1: Putting everything into the same coordinate frame as $B(u,v)$.
  – $x(u,v), y(u,v), z(u,v)$ – this is given for parametric surfaces, but easy to derive for other analytical surfaces.
  – Or $\mathbf{O}(u,v)$
Bump Mapping

- Define the tangent plane to the surface at a point \((u,v)\) by using the two vectors \(O_u\) and \(O_v\).
- The normal is then given by:
  - \(N = O_u \times O_v\)
Bump Mapping

• The new surface positions are then given by:
  • \( O'(u,v) = O(u,v) + B(u,v) \mathbf{N} \)
  • Where, \( \mathbf{N} = \mathbf{N} / |\mathbf{N}| \)

• Differentiating leads to:
  • \( O'_u = O_u + B_u \mathbf{N} + B (\mathbf{N})_u \approx O'_u = O_u + B_u \mathbf{N} \)
  • \( O'_v = O_v + B_v \mathbf{N} + B (\mathbf{N})_v \approx O'_v = O_v + B_v \mathbf{N} \)

If \( B \) is small.
Bump Mapping

- This leads to a new normal:

\[ N'(u,v) = O_u \times O_v - B_u(N \times O_v) + B_v(N \times O_u) + B_u B_v(N \times N) \]

\[ \Rightarrow = N - B_u(N \times O_v) + B_v(N \times O_u) \]

\[ \Rightarrow = N + D \]
Bump Mapping

- For efficiency, can store $B_u$ and $B_v$ in a 2-component texture map.
- The cross products are geometry terms only.
- $N'$ will of course need to be normalized after the calculation and before lighting.
  - This floating point square root and division makes it difficult to embed into hardware.
Displacement Mapping

- Modifies the surface position in the direction of the surface normal.
Displacement Mapping

• Bump mapping has a limitation on how much you can tweak
• If the desired amount of change is too large for bump mapping, can use displacement mapping.
• Actually go and modify the surface geometry, and re-calculate the normals
• Quite expensive
Environment Mapping

- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray: \( R = 2(N \cdot V)N - V \)
- What is in the map?
Approximations Made

• The map should contain a view of the world with the point of interest on the object as the eye
  – We can’t store a separate map for each point, so one map is used with the eye at the center of the object
  – Introduces distortions in the reflection, but the eye doesn’t notice
  – Distortions are minimized for a small object in a large room
• The object will not reflect itself
• The mapping can be computed at each pixel, or only at the vertices
Environment Maps

- The environment map may take one of several forms:
  - Cubic mapping
  - Spherical mapping (two variants)
  - Parabolic mapping
- Describes the shape of the surface on which the map “resides”
- Determines how the map is generated and how it is indexed
- What are some of the issues in choosing the map?
Example
Cubic Mapping

• The map resides on the surfaces of a cube around the object
  – Typically, align the faces of the cube with the coordinate axes

• To generate the map:
  – For each face of the cube, render the world from the center of the object with the cube face as the image plane
    • Rendering can be arbitrarily complex (it’s off-line)
  – Or, take 6 photos of a real environment with a camera in the object’s position
    • Actually, take many more photos from different places the object might be
    • Warp them to approximate map for all intermediate points

• Remember *The Abyss* and *Terminator 2*?
Cubic Map Example
Indexing Cubic Maps

• Assume you have $R$ and the cube’s faces are aligned with the coordinate axes, and have texture coordinates in $[0,1]x[0,1]$
  – How do you decide which face to use?
  – How do you decide which texture coordinates to use?

• What is the problem using cubic maps when texture coordinates are only computed at vertices?
Lat/Long Mapping

- The original algorithm (1976) placed the map on a sphere centered on the object
- Mapping functions assume that $s, t$ equate to latitude and longitude on the sphere:
  \[ s = \frac{1}{2} \left( 1 + \frac{1}{\pi} \tan^{-1} \left( \frac{R_y}{R_x} \right) \right), \quad t = \frac{R_z + 1}{2} \]
- What is bad about this method?
  - Sampling
  - Map generation
  - Complex texture coordinate computations
Sphere Mapping

• Again the map lives on a sphere, but now the coordinate mapping is simplified

• To generate the map:
  – Take a map point \((s, t)\), cast a ray onto a sphere in the \(-Z\) direction, and record what is reflected
  – Equivalent to photographing a reflective sphere with an orthographic camera (long lens, big distance)
    • Again, makes the method suitable for film special effects
A Sphere Map
Indexing Sphere Maps

- Given the reflection vector:
  
  \[ s = \frac{R_x}{m} + \frac{1}{2}, \quad t = \frac{R_y}{m} + \frac{1}{2} \]

  \[ m = 2\left(R_x^2 + R_y^2 + (R_z + 1)^2\right)^\frac{1}{2} \]

- Implemented in hardware

- Problems:
  - Highly non-uniform sampling
  - Highly non-linear mapping
Example
Parabolic Mapping

- Assume the map resides on a parabolic surface
  - Two surfaces, facing each other
- Improves on sphere maps:
  - Texture coordinate generation is a near linear process
  - Sampling is more uniform
  - Result is more view-independent
- However, requires multi-passes to implement, so not generally used
Partially Reflective Objects

- Use multi-texturing hardware
  - First stage applied color texture
  - Second stage does environment mapping using alpha blend with existing color