CS 378: Computer Game Technology

Texture, Bump, Light, Environment Maps
Spring 2012
Texture Mapping

The problem: Colors, normals, etc. are only specified at vertices. How do we add detail between vertices?

Solution: Specify the details in an image (the texture) and specify how to apply the image to the geometry (the map)

Works for shading parameters other than color, as we shall see

The basic underlying idea is the mapping
Basic Mapping

- The texture lives in a 2D space
  - Parameterize points in the texture with 2 coordinates: (s,t)
- Define the mapping from (x,y,z) in world space to (s,t) in texture space
- With polygons:
  - Specify (s,t) coordinates at vertices
  - Interpolate (s,t) for other points based on given vertices
I assume you recall...

- Texture sampling (aliasing) is a big problem
  - Mipmaps and other filtering techniques are the solution
- The texture value for points that map outside the texture image can be generated in various ways
  - Repeat, Clamp, …
- Texture coordinates are specified at vertices and interpolated across triangles
- Width and height of texture images is constrained (powers of two, sometimes must be square)
Multitexturing

- Some effects are easier to implement if multiple textures can be applied
  - Future lectures: Light maps, bump maps, shadows, …
- Multitexturing hardware provides a pipeline of texture units, each of which applies a standard texture map operation
  - Fragments are passed through the pipeline with each step working on the result of the previous stage
  - Texture parameters are specified independently for each unit, further improving functionality
  - For example, the first stage applies a color map, the next modifies the illumination to simulate bumps, the third modifies opacity
  - Not the same as multi-pass rendering - all applied in one pass
Pixel Shaders

- Current generation hardware provides pixel shaders
- A pixel shader operates on a fragment
  - A single pixel (or sub-pixel) that has already been “lit”
- It can compute texture coordinates, do general texture look-ups, modify color/depth/opacity, and some other functions
- More general than multi-texturing and very powerful
Textures in Games

- The game engine provides some amount of texture support
- Artists are supplied with tools to exploit this support
  - They design the texture images
  - They specify how to apply the image to the object
- Commonly, textures are supplied at varying resolutions to support different hardware performance
  - Note that the texture mapping code does not need to be changed - just load different sized maps at run time
- Textures are, without doubt, the most important part of a game’s look
Example Texture Tool
Packaging Textures

- **Problem:** The limits on texture width/height make it inefficient to store many textures
  - For example: long, thin objects
- **Solution:** Artists pack the textures for many objects into one image
  - The texture coordinates for a given object may only index into a small part of the image
  - Care must be taken at the boundary between sub-images to achieve correct blending
  - Mipmapping is restricted
  - Best for objects that will be at known resolution (weapons, for instance)
Combining Textures
Texture Matrix

- Normally, the texture coordinates given at vertices are interpolated and directly used to index the texture.
- The texture matrix applies a homogeneous transform to the texture coordinates before indexing the texture.
- What use is this?
Animating Texture (method 1)

- Loading a texture onto the graphics card is very expensive.
- But once there, the texture matrix can be used to "transform" the texture.
  - For example, changing the translation can select different parts of the texture.
- If the texture matrix is changed from frame to frame, the texture will appear to move on the object.
- This is particularly useful for things like flame, or swirling vortices, or pulsing entrances, …
Projective Texturing

- The texture should appear to be projected onto the scene, as if from a slide projector

Solution:
- Equate texture coordinates with world coordinates
- Think about it from the projector’s point of view: wherever a world point appears in the projector’s view, it should pick up the texture
- Use a texture matrix equivalent to the projection matrix for the projector – maps world points into texture image points

Details available in many places

Problems? What else could you do with it?
What’s in a Texture?

- The graphics hardware doesn’t know what is in a texture
  - It applies a set of operations using values it finds in the texture, the existing value of the fragment (pixel), and maybe another color
  - The programmer gets to decide what the operations are, within some set of choices provided by the hardware

Examples:
- The texture may contain scalar “luminance” information, which simply multiplies the fragment color. What use is this?
- The texture might contain “alpha” data that multiplies the fragment’s alpha channel but leaves the fragment color alone. What use is this?
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
Cube 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?
Cube 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 \([0,1] \times [0,1]\)
  - 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?