Advanced texture mapping

Computer Graphics
COMP 770 (236)
Spring 2007

Instructor: Brandon Lloyd
From last time...

- Physically based illumination models
- Cook-Torrance illumination model
  - Microfacets
  - Geometry term
  - Fresnel reflection
- Radiance and irradiance
- BRDFs
Today’s topics

- Texture coordinates
- Uses of texture maps
  - reflectance and other surface parameters
  - lighting
  - geometry
- Solid textures
Uses of texture maps

- Texture maps are used to add complexity to a scene
- Easier to paint or capture an image than geometry
- Model reflectance
  - attach a texture map to a parameter
- Model light
  - environment maps
  - light maps
- Model geometry
  - bump maps
  - normal maps
  - displacement maps
  - opacity maps and billboards
Specifying texture coordinates

- Texture coordinates needed at every vertex
- Hard to specify by hand
- Difficult to wrap a 2D texture around a 3D object

from Physically-based Rendering
Planar mapping

- Compute texture coordinates at each vertex by projecting the map coordinates onto the model
Cylindrical mapping
Spherical mapping

2/21/07
Cube mapping
“Unwrapping” the model

images from [www.eurecom.fr/~image/Clonage/geometric2.html](http://www.eurecom.fr/~image/Clonage/geometric2.html)
Modelling surface properties

- Can use a texture to supply any parameter of the illumination model
  - ambient, diffuse, and specular color
  - specular exponent
  - roughness

from www.ronfrazier.net
Modelling lighting

- **Light maps**
  - supply the lighting directly
  - good for static environments

- **Projective textures**
  - can be used to simulate a spot light
  - shadow maps

- **Environment maps**
  - A representation of the scene around an object
  - Good for reflection and refraction
Light maps in Quake

- Light maps are used to store pre-computed illumination
- Texture maps add detail to surfaces

<table>
<thead>
<tr>
<th>Texture Maps</th>
<th>Light Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>RGB</td>
</tr>
<tr>
<td>Instanced</td>
<td>Yes</td>
</tr>
<tr>
<td>Resolution</td>
<td>High</td>
</tr>
</tbody>
</table>

Light map image by Nick Chirkov
Projective textures

- Treat the texture as a slide in a projector
- No need to specify texture coordinates explicitly
- A good model for shading variations due to illumination (cool spotlights)
- Projectors work like cameras in reverse
  - camera: color of point in scene \(\rightarrow\) color of corresponding pixel
  - projector: color of pixel \(\rightarrow\) color of corresponding point in the scene
OpenGL TexGen

- OpenGL can generate texture coordinates directly from the vertices
- Each component of a texture coordinate is generated by taking the dot product of the corresponding vertex with a specified plane
- In EYE-LINEAR mode, the planes are multiplied by the inverse of the current modelview matrix when they are specified

\[
\begin{bmatrix}
  r \\
  s \\
  t \\
  q
\end{bmatrix} = \begin{bmatrix}
  s \\
  t \\
  r \\
  q
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

\[
MV^{-1}
\]
OpenGL TexGen

- Setting the texgen planes to the identity matrix while the view matrix sits on the top of the stack results in texture coordinates that correspond to world coordinates.

```c
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR)
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR)
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR)
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR)

# Assumes that view matrix is on the stack

eyePlaneS = [ 1.0, 0.0, 0.0, 0.0 ]
eyePlaneT = [ 0.0, 1.0, 0.0, 0.0 ]
eyePlaneR = [ 0.0, 0.0, 1.0, 0.0 ]
eyePlaneQ = [ 0.0, 0.0, 0.0, 1.0 ]

glTexGenfv(GL_S, GL_EYE_PLANE, eyePlaneS)
glTexGenfv(GL_T, GL_EYE_PLANE, eyePlaneT)
glTexGenfv(GL_R, GL_EYE_PLANE, eyePlaneR)
glTexGenfv(GL_Q, GL_EYE_PLANE, eyePlaneQ)
```
OpenGL TexGen

- TexGen must be enabled/disabled

```python
if (projTexture):
    glEnable(GL_TEXTURE_GEN_S)
    glEnable(GL_TEXTURE_GEN_T)
    glEnable(GL_TEXTURE_GEN_R)
    glEnable(GL_TEXTURE_GEN_Q)
...

# draw scene
if (projTexture):
    glDisable(GL_TEXTURE_GEN_S)
    glDisable(GL_TEXTURE_GEN_T)
    glDisable(GL_TEXTURE_GEN_R)
    glDisable(GL_TEXTURE_GEN_Q)
    ...
```
Texture space

- Texture coordinates can be modified by the GL_TEXTURE matrix.
- The transform we want to apply to world coordinates is:

\[
T = \begin{bmatrix}
\frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & \frac{1}{2} & 0 & \frac{1}{2} \\
0 & 0 & \frac{1}{2} & \frac{1}{2} \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
P_{pr} \\
V_{pr}
\end{bmatrix}
\]

- We multiply by the projection and view matrices of the projector just as we do with a camera.
- Then we multiply by a matrix that goes from NDC to the [0,1) range used by texture coordinates.
OpenGL example

The following piece of code sets up the desired texture matrix

```c
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glTranslated(0.5, 0.5, 0.5); // Scale and bias the [-1,1] NDC values
glScaled(0.5, 0.5, 0.5);     // to the [0,1] range of the texture map
gluPerspective(15, 1, 5, 7); // projector's "projection" and view matrices
gluLookAt(lightPosition[0],lightPosition[1],lightPosition[2], 0,0,0, 0,1,0);
glMatrixMode(GL_MODELVIEW);
```
What about that r coordinate?

- Texture coordinates are homogeneous 4-tuples
- The q coordinate is the homogeneous coordinate
  - the texture coordinate is divided by q before the texture lookup
- The r coordinate as we are using it represents depth
- Can be used to compute shadows
Shadow maps

Use the depth map in the light view to determine if sample point is visible.

Point in shadow visible to the eye by not visible to the light.
Algorithm:

- Render scene from the light’s view
- Save depth buffer in a texture
- Set up texgen for projective texturing
- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_R_TO_TEXTURE)`
- Bind the depth texture
- Render the scene
Environment maps

- Can use transformed surface normal to compute texture coordinates
- This kind of mapping can be used to simulate reflections, and other shading effects.
- Not completely accurate
  - Assumes that all reflected rays begin from the same point
  - Assumes that all objects in the scene are the same distance from that point.
Sphere mapping basics

- Sphere mapping maps the normals of the object to the corresponding normal of a sphere. It uses a texture map of a sphere viewed from infinity to establish the color for the normal.
Sphere mapping

- Mapping the normal to a point on the sphere

Recall:

\[ \vec{r} = 2(\vec{n} \cdot \vec{v})\vec{n} - \vec{v} \]

\[ \alpha \vec{n} = \vec{r} + \vec{v} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \frac{\alpha \vec{n}}{||\alpha \vec{n}||} = \begin{bmatrix} \frac{r_x}{p} \\ \frac{r_y}{p} \\ \frac{r_z+1}{p} \end{bmatrix} \]

\[ p = \sqrt{r_x^2 + r_y^2 + (r_z+1)^2} \]

The normal on a sphere corresponding to the computed reflection vector and a viewer infinitely far away.

\[ s = \frac{r_x}{p} \quad t = \frac{r_y}{p} \]

\[ s' = \frac{s}{2} + \frac{1}{2} \quad t' = \frac{t}{2} + \frac{1}{2} \]

\[ s' = \frac{r_x}{2p} + \frac{1}{2} \quad t' = \frac{r_y}{2p} + \frac{1}{2} \]
OpenGL example

This was a very special purpose hack in OpenGL, however, we have it to thank for a lot of the flexibility in today’s graphics hardware… this hack was the genesis of programmable vertex shading.
What’s the best map?

A sphere map is not the only representation choice for environment maps. There are alternatives, with more uniform sampling properties, but they require different normal-to-texture mapping functions.
Problems with environment maps

- Expensive to update dynamically
- May not represent the “near field” very well
- Subject to errors like the one shown below

![Reflection of swimming pool is wrong](images_from_NVIDIA)
The main idea is to store complex surface details in a texture rather than modeling them explicitly.

Bump maps
- modify the existing normal

Normal maps
- replace the existing normal

Displacement maps
- modify the geometry

Opacity maps and billboards
- knock-out portions of a polygon using the alpha channel
Bump mapping

- Modifies the normal not the actual geometry
- Texture treated as a heightfield
- Partial derivatives used to change the normal
- Causes surface to appear deformed by the heightfield
More bump map examples

- Silhouette edge of the object not affected.
- Assumes that the illumination model is applied at every pixel
One more bump map example
Normal mapping

- Replaces the normal rather than tweaking it

original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh and normal mapping
500 triangles
Normal mapping

- A normal map for a low resolution model can be created automatically from a high resolution model

Displacement mapping

- Texture maps can be used to actually move surface points.

- The geometry must be displaced before visibility is determined
  - Is this easily done in the graphics pipeline?
Opacity maps

Use the alpha channel to make portions of the texture transparent. Cheaper than explicit modelling.
Billboards

Replace complex geometry with polygons texture mapped with transparent textures
3D or solid textures

- Solid textures are three dimensional assigning values to points in 3 space
- Very effective at representing some types of materials such as marble and wood.
- The object is “carved” out of the solid texture
- Generally, solid textures are defined procedural functions rather than tabularized or sampled functions as used in 2-D
  - Why?
- We will explore an approach based on: *An Image Synthesizer*, by Ken Perlin, SIGGRAPH '85. The vase to the right is from this paper.
Noise and turbulence are two primitive functions which can be used to build up visually-rich, solid textures. They add random variations in a controlled way.

A *noise function* is a continuous function that varies throughout space at a uniform frequency. To create a simple noise function, consider a 3D lattice, with a random value assigned to each integral coordinate:
To calculate the noise value of any point in space, we first determine which cube of the lattice the point is in. Next, we interpolate the desired value using the 8 corners of the cube:

- Trilinear interpolation is illustrated above
- Higher-order interpolation can also be used.
Evaluating noise

- Since noise is a 3D function, we can evaluate it at any point we want
  - We don't have to worry about mapping the noise to the object
  - Just use the (x, y, z) coordinate of each point as our 3D texture coordinate
  - Has the appearance of carving an object out of a big block of noise
Turbulence

- Turbulence adds details at different scales by summing different frequencies of noise
- We get higher frequency noise simply by scaling the input coordinate, i.e. noise(2p) has twice the frequency of noise(p)
Marble example

\[
\sin \left( x + \sum_{i=1}^{\frac{1}{2}} \text{noise}(2^i p) \right)
\]

This value used as index into look up table consisting of stripes of varying thickness.

\[
\sum_{i=1}^{\frac{1}{2}} \text{noise}(2^i p)
\]

image from http://www.noisemachine.com/talk1/19.html
Next time

- Illumination and shading - III
  - Render cheats and hacks
- Review HW 1 and Quiz 1