

Advanced Texture Mapping

- Difficulties with texture mapping
- Projective Texturing
- Shadow Mapping
- Environment Mapping
- Multi-texturing


[Lecture 20](#)

Slide 1

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide01.html> [12/2/2000 12:48:15 AM]

Review of *Label* Textures

- Increases the apparent complexity of simple geometry
- Must specify texture coordinates for each vertex
- Projective correction (can't linearly interpolate in screen space)
- Specify variations in shading within a primitive
- Two aspects of shading
 - Illumination
 - Surface Reflectance
- Label textures can handle both kinds of shading effects but it gets tedious
- Acquiring label textures is surprisingly tough


[Lecture 20](#)

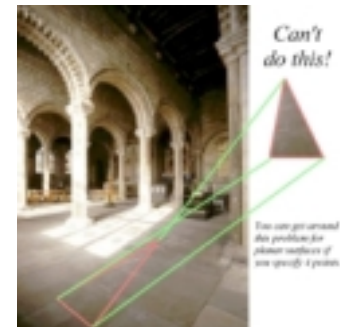
Slide 2

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide02.html> [12/2/2000 12:48:20 AM]

Difficulties with Label Textures

- Tedious to specify texture coordinates for every triangle
- Textures are attached to the geometry
- Easier to model variations in reflectance than illumination
- Can't use just any image as a label texture
The "texture" can't have projective distortions
 Reminder: linear interpolation in image space is not equivalent to linear interpolation in 3-space (This is why we need "perspective-correct" texturing). The converse is also true.
- Textures are attached to the geometry
- Easier to model variations in reflectance than illumination
- Makes it hard to use pictures as textures

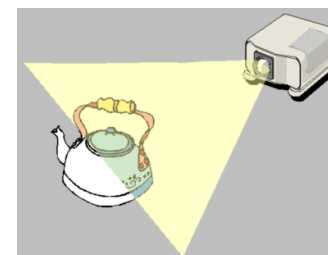

[Lecture 20](#)

Slide 3

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide03.html> [12/2/2000 12:48:27 AM]

Projective Textures



- Treat the texture as a light source (like a slide projector)
- No need to specify texture coordinates explicitly
- A good model for shading variations due to illumination
- A fair model for reflectance (can use pictures)

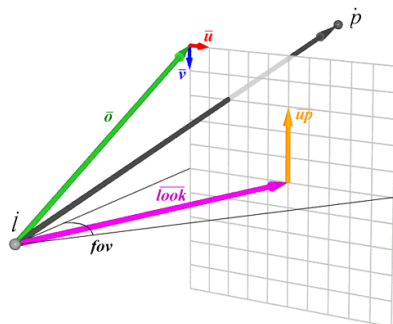

[Lecture 20](#)

Slide 4

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide04.html> [12/2/2000 12:48:29 AM]

Projector Geometry



$$\dot{p} = \dot{i} + t \underbrace{\begin{bmatrix} u_x & v_x & o_x \\ u_y & v_y & o_y \\ u_z & v_z & o_z \end{bmatrix}}_{\mathbf{P}} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

$$t \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} = \mathbf{P}^{-1}(\dot{p} - \dot{i})$$

(See last lecture for details on how to compute the \mathbf{P} matrix)



Lecture 20

Slide 5

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide05.html> [12/2/2000 12:48:33 AM]

The Mapping Process

During the Illumination process:

For each vertex of triangle

(in world or lighting space)

Compute ray from the projective texture's origin to point

Compute homogeneous texture coordinate, $[t_i, t_j, t]$

(use equation from last slide)

During scan conversion

(in projected screen space)

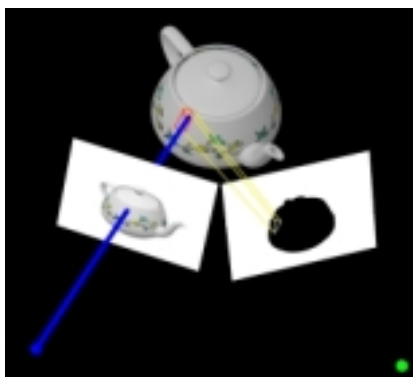
Interpolate all three texture coordinates in 3-space

(premultiply by w of vertex)

Do normalization at each rendered pixel

$$i = t_i / t, \quad j = t_j / t$$

Access projected texture



Lecture 20

Slide 6

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide06.html> [12/2/2000 12:48:36 AM]

Projective Texture Mapping as a Light Source

```
public Light(float px, float py, float pz, float lx, float ly, float lz,
            float upx, float upy, float upz, float hfov, Raster r) {
    lightType = SHADER;
    float t, ux, uy, uz, vx, vy, vz;

    x = px; y = py; z = pz;
    ir = 0; ig = 0; ib = 0;

    lx = lx - x; ly = ly - y; lz = lz - z;
    t = (float)(1 / Math.sqrt(lx*lx + ly*ly + lz*lz));
    lx *= t; ly *= t; lz *= t;

    ux = ly*upz - lz*upy; uy = lz*upx - lx*upz; uz = lx*upy - ly*upx;
    t = (float)(1 / Math.sqrt(ux*ux + uy*uy + uz*uz));
    ux *= t; uy *= t; uz *= t;

    vx = ly*uz - lz*uy; vy = lz*ux - lx*uz; vz = lx*uy - ly*ux;
    t = (float)(1 / Math.sqrt(vx*vx + vy*vy + vz*vz));
    vx *= t; vy *= t; vz *= t;

    t = (float)(1 / (2*Math.tan((0.5*hfov)*Math.PI/180)));
    lx = lx*t - 0.5f*(ux + vx);
    ly = ly*t - 0.5f*(uy + vy);
    lz = lz*t - 0.5f*(uz + vz);

    setProjective(lx, ly, lz, ux, uy, uz, vx, vy, vz); // Inverts matrix
    lightTexture = r;
}
```



Lecture 20

Slide 7

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide07.html> [12/2/2000 12:48:38 AM]

A Little More Code

In Triangle.Illuminate() ...

```
if (l[i].lightType == Light.SHADER) {
    lx = vlist[v[j]].x - l[i].x;
    ly = vlist[v[j]].y - l[i].y;
    lz = vlist[v[j]].z - l[i].z;

    float u, v, w;
    u = l[i].m[0]*lx + l[i].m[1]*ly + l[i].m[2]*lz;
    v = l[i].m[3]*lx + l[i].m[4]*ly + l[i].m[5]*lz;
    w = l[i].m[6]*lx + l[i].m[7]*ly + l[i].m[8]*lz;
    tq[j] = (int) (lightTexture.width*TSCALE*u);
    ts[j] = (int) (lightTexture.height*TSCALE*v);
    tt[j] = (int) (TSCALE*w);
} else
```

In Triangle.ScanConvert() after setting up plane equations for q, s, and t...

```
if (lightTexture != null) {
    int i = (int) (q/t);
    int j = (int) (s/t);
    if (i < 0) i = 0;
    else if (i >= lightTexture.width) i = lightTexture.width - 1;
    if (j < 0) j = 0;
    else if (j >= lightTexture.height) j = lightTexture.height - 1;
    int rgb = lightTexture.getPixel(i, j);
}
```



Lecture 20

Slide 8

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide08.html> [12/2/2000 12:48:39 AM]

Projective Texture Examples

First, let's consider projective textures as a source of illumination...

These are the two textures that were used:



and


[Lecture 20](#)

Slide 9

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide09.html> [12/2/2000 12:48:41 AM]

More Examples

First, let's consider projective textures to model reflectance...

The texture used was



. Since we are viewing the scene from a slightly different

point of view than the projective texture we see some points that are not shaded.


[Lecture 20](#)

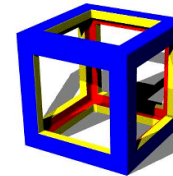
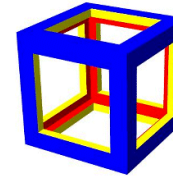
Slide 10

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide10.html> [12/2/2000 12:48:42 AM]

Shadow Maps

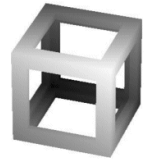
Projective Textures with Depth



Textures can also be used to generate shadows. First, the scene is rendered from the point of view of each light source, but only the depth-buffer values are retained.

In this example the closer points are lighter and more distant parts are darker (with the exception of the most distant value which is shown as white for contrast).

As each pixel is shaded (once more shadow mapping assumes that the illumination model is applied at each pixel) a vector from the visible point to the light source is computed (Remember it is needed to compute, $N \cdot L$). As part of normalizing it we compute its length. If we find the projection of the 3D point that we are shading onto each light's shadow buffer we can compare this length with the value stored in the shadow buffer. If the shadow-buffer is less than the current point's length then the point is in shadow and the corresponding light source can be ignored for that point.


[Lecture 20](#)

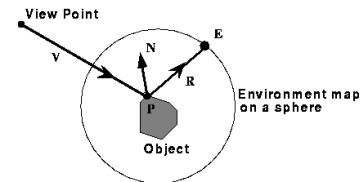
Slide 11

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide11.html> [12/2/2000 12:48:44 AM]

Environment Maps

If, instead of using the ray from the surface point to the projected texture's center, we used the *direction* of the reflected ray to index a texture map. We can simulate reflections. This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.

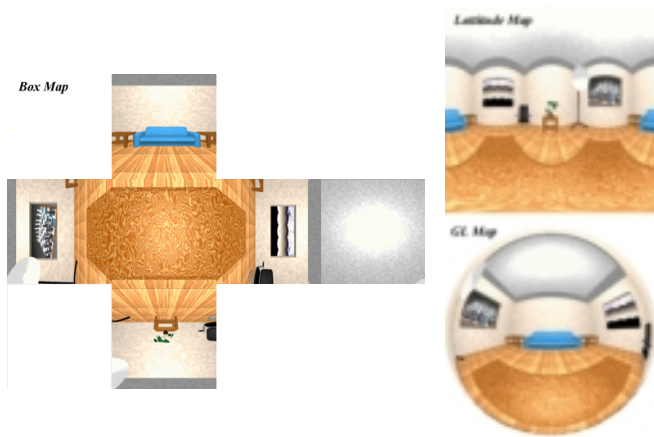

[Lecture 20](#)

Slide 12

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide12.html> [12/2/2000 12:48:45 AM]

What's the Best Chart?


[Lecture 20](#)

Slide 13

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide13.html> [12/2/2000 12:48:47 AM]

Other Texture Mappings

A small variation on environment maps- specular maps.

Specular maps can model extended or areas light sources in the scene. This can be done in combination with modeling the environment. This approach gives a much smoother highlight when per-vertex shading is used to compute specular highlights.


[Lecture 20](#)

Slide 14

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide14.html> [12/2/2000 12:48:49 AM]

Relief Textures

Texture Maps with Depth...


[Lecture 20](#)

Slide 15

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide15.html> [12/2/2000 12:48:50 AM]

The Best of All Worlds

All these texture mapping modes are great! The problem is, no one of them does everything well. Suppose we allowed several textures to be applied to each primitive during rasterization.


[Lecture 20](#)

Slide 16

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide16.html> [12/2/2000 12:48:52 AM]

Multipass vs. Multitexture

Multipass (the old way) - Render the image in multiple passes, and "add" the results.

Multitexture - Make multiple texture accesses within the rasterizing loop and "blend" results.

Blending approaches:

- Texture modulation
- Alpha Attenuation
- Additive textures
- Weird modes


[Lecture 20](#)

Slide 17

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide17.html> [12/2/2000 12:48:53 AM]

Texture Mapping in Quake

Quake uses *light maps* in addition to texture maps. Texture maps are used to add detail to surfaces, and light maps are used to store pre-computed illumination. The two are multiplied together at run-time, and cached for efficiency.

Textures Only

Textures & Light Maps

	Texture Maps	Light Maps
Data	RGB	Intensity
Instanced	Yes	No
Resolution	High	Low

Light map image
by Nick Chirkov.


[Lecture 20](#)

Slide 18

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide18.html> [12/2/2000 12:48:55 AM]

Examples

Next time... Texture mapping techniques with per-pixel shading.


[Lecture 20](#)

Slide 19

6.837 Fall '00

<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture20/Slide19.html> [12/2/2000 12:48:57 AM]