

Projection texture mapping

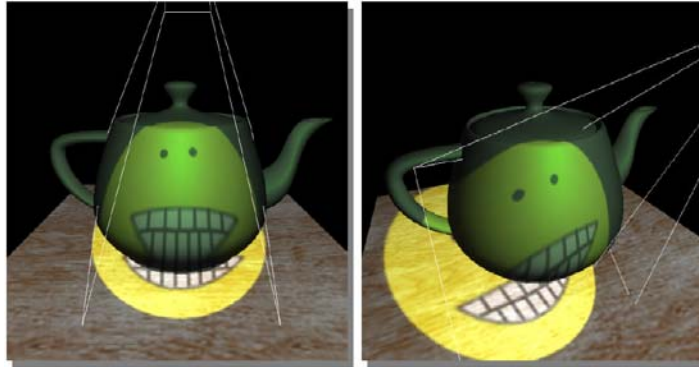
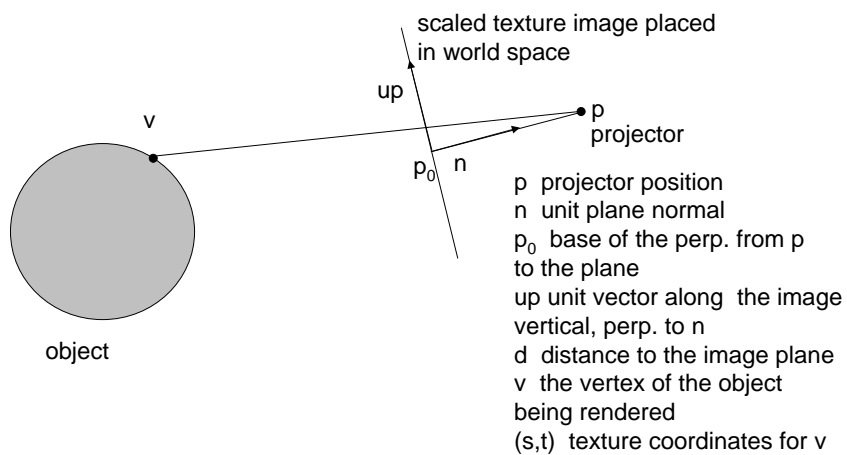


image from Case Everitt notes

Project texture to objects as if it was a slide projection; achieved either by computing texture coordinates explicitly or using OpenGL texture coordinate generation

Projection texture mapping

Computing texture coordinates in your code



Projection texture mapping

Computing texture coordinates in your code:

- compute P_v , the projection of v to the plane in world coordinates : $p + d(v-p)/\text{dot}(v-p,n)$
- convert P_v to plane coordinates in the coord. system $(n \times up, up)$ with origin at p_0 to get (s,t)
- if necessary rescale s and t to give the image the desired size on the object

Projection texture mapping

Using OpenGL

- OpenGL does not provide a simple way to specify the natural parameters (projector position etc)
- what it provides is ability to compute texture coordinates automatically, using linear equations applied to vertex coordinates either **object** (before any transforms are applied) or **eye (after the modelview transform is applied)**. `glTexGen` function is used to specify relevant parameters.

Projection texture mapping

Using OpenGL, single texture coordinates

- tell OpenGL we are using texture generation in object coords for S

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR;
```

- setup the equation for computing the S coord

```
float [] planeS = { 1.0f, 0.0f, 0.0f, 0.0f };  
glTexGenfv(GL_S, GL_OBJECT_PLANE, planeS);
```

- before drawing the objects make sure the texture coord generation is enabled and disable it after:

```
glEnable(GL_TEXTURE_2D);  
glEnable(GL_TEXTURE_GEN_S);  
...  
glDisable(GL_TEXTURE_GEN_S);
```

Projection texture mapping

The previous code results in texture coordinate s computed automatically as $S = 1*x + 0*y + 0*z + 1 = x + 1$ for a vertex (x,y,z) (in **object** coords)

Similarly, you can enable T computed as

$$T = 0*x + 1*y + 0*z + 1 = y + 1$$

This is not very interesting – you may as well use `TexCoord2(x+1,y+1)`

But using different planes for S and T allows you to change the projection direction; you can think about S and T as being (scaled) distance to the plane with equation

$$x_0*x + y_0*y + z_0*z + w_0 = 0, \text{ with } w_0 \text{ setting the scale.}$$

The two planes need to be parallel to the projection direction

Projective texture mapping

There are two fundamental limitations to the prev. approach:

- the projection is orthographics, i.e. the projector is at infinity;
- it is attached to an object rather than fixed in space, i.e. any modeling transformation would also apply to the object

To get rid of these limitations, one needs to fully model perspective transformation:

- use 4 texture coordinates (s,t,r,q) and texture transformations;
- eye space calculation (i.e. replace GL_OBJECT_LINEAR with EYE_LINEAR), or tracking of the modeling (not to be confused with combined modelview) matrix is needed