Q: Why do we need texture mapping?
A: Because objects look fake and boring without it.

Adding texture improves realism.

We would like to map the texture image coordinates to surface coordinates.

We do so, we use an intermediate map to normalized coordinates.
What is texture mapping?

for each pixel in the image projection of the polygon{
  compute corresponding texel position
  copy texture RGB to image pixel RGB
}

Let's think about the matrices that are used for this mapping. Here we simplify: assume camera coords = world coords.

Homography

A homography is an invertible 3x3 matrix that maps between 2D spaces that are represented in homogeneous coordinates.

for each pixel in the image projection of the polygon{
  Use (inverse) homography to compute corresponding texel position
  Use texture RGB to determine image pixel RGB
}

Details

How to construct the homography?

What are the sampling issues and how to deal with them?
This is the usual projection from world coordinates to clip coordinates, but now the z row has been deleted.

Why? Because we can ignore hidden surface removal in this mapping.

[ADDED April 25: I have ignored the normalization transformation in this example. This is only allowed in the special case that the normalization matrix is the identity matrix.]

Example: slanted floor

Take $z = 0$ plane, rotate it by $\theta$ degrees around x axis, and translate it by $z_0$.

$$\begin{align*}
z &= z_0 - y \tan \theta \\
z &= (z_0, 0, 0)
\end{align*}$$

Calculating $H$ and its inverse gives....

$$H = \begin{bmatrix} f & 0 & 0 \\ 0 & f \cos \theta & 0 \\ 0 & -f \sin \theta & 1 \end{bmatrix}$$

$$H^{-1} = \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f \cos \theta & 0 \\ 0 & 0 & 1/\cos \theta
\end{bmatrix}$$

Exercise:

$H$ maps which points in $\mathbb{R}^2$ to points at infinity (2D)?

$H$ maps points at infinity to which image points?

OpenGL computes the homographies for you.

Details

How to construct the homography?

What are the sampling issues and how to deal with them?

Exercise: where in the pipeline does this occur?
**Texture magnification:** a pixel in texture image ('texel') maps to an area larger than one pixel in image

\[ I(x_p, y_p) \rightarrow T(s_p, t_p) \]

**Texture minification:** a pixel in texture image ('texel') maps to an area smaller than a pixel in image

\[ I(x_p, y_p) \rightarrow T(s_p, t_p) \]

- **Details**
  - **How to construct the homography?**
    - Q: What are the sampling issues ...
    - A: "Aliasing"
    - Q: ... and how to deal with them ?
    - A: "Anti-aliasing"

I will just give a sketch. A proper treatment would take several weeks.

"Aliasing" in computer graphics:
For any RGB image defined on a discrete grid of pixels, there are infinitely many images defined on the 2D continuum, that have the same RGB values at the discrete pixels.

"Aliasing" in programming languages:
Two variables reference the same memory location.

\[ x = \text{new\ Dog()} \]
\[ y = x \]

**Textures ("regular") and aliasing**

Suppose we sample the stripe image on the left using the intersection points (pixels) on the grid on the right.

- **Q:** Will we also get regular vertical stripes in the sampled image?
- **A:** No, unless the distance between pixels happens to correspond exactly to the stripe width.
e.g. Moiré patterns
Caused by camera pixel frequency being higher than that of the grid pattern on the big central door (minification).

Somehow this slide was dropped from the lecture...
Too bad, because it’s a classic.

from Paul Heckbert, “Survey of Texture Mapping”

How to reduce aliasing in texture mapping?

How to choose \( I( x_p, y_p) \)?

Change in notation for upcoming slides
Recall that a pixel can be thought of in two ways: as a little square, and as a point with integer coordinates. Texture mapping is a good example of why we need this flexibility. Up to now in this lecture, the pixels have been little squares. But in the following few slides, pixels in the texture image will defined as a grid of points, namely the intersections of the horizontal and vertical lines of the grid. In particular, each square in the texture grid is no longer a pixel. Rather, the corner points of the square are the pixels.

This should make more sense once you see the arguments.

Solution 1a: Linear interpolation
Partition square into two triangles. Use linear interpolation within the triangle that \((x, y)\) lies.

Solution 1b: "Bilinear interpolation"
Exercise: Write out the formula for \(T(x, y)\)

Case 1: magnification

Case 2: texture minification

Solution: (not used by OpenGL) Take average of intensities within the quad (inverse map of square pixel).
OpenGL used MIP mapping (next lecture).
Here is an example of what these two solutions can produce. For minification, averaging produces grey pixels, which is appropriate. For magnification, the interpolation blurs the intensities, which is in this case doesn’t work so well because there is too much of it. (We only wanted to blur the edge enough to hide the jaggies!)

How to texture map a quadric (or bicubic)?

Discretize sphere into polygons (or use parametric surface patch).

Texture mapping in OpenGL (what is required?)

- a texture image
  
gTexImage2D(GL_TEXTURE_2D, ..... , size, .... , data )
- a correspondence between polygon vertices and texture coordinates
  
gBegin(GL_POLYGON)
gTexCoord2f(0, 0);
gVertex( ..... )
gTexCoord2f(0, 1);
gVertex( ..... )
gTexCoord2f(1, 0);
gVertex( ..... )
gEnd()