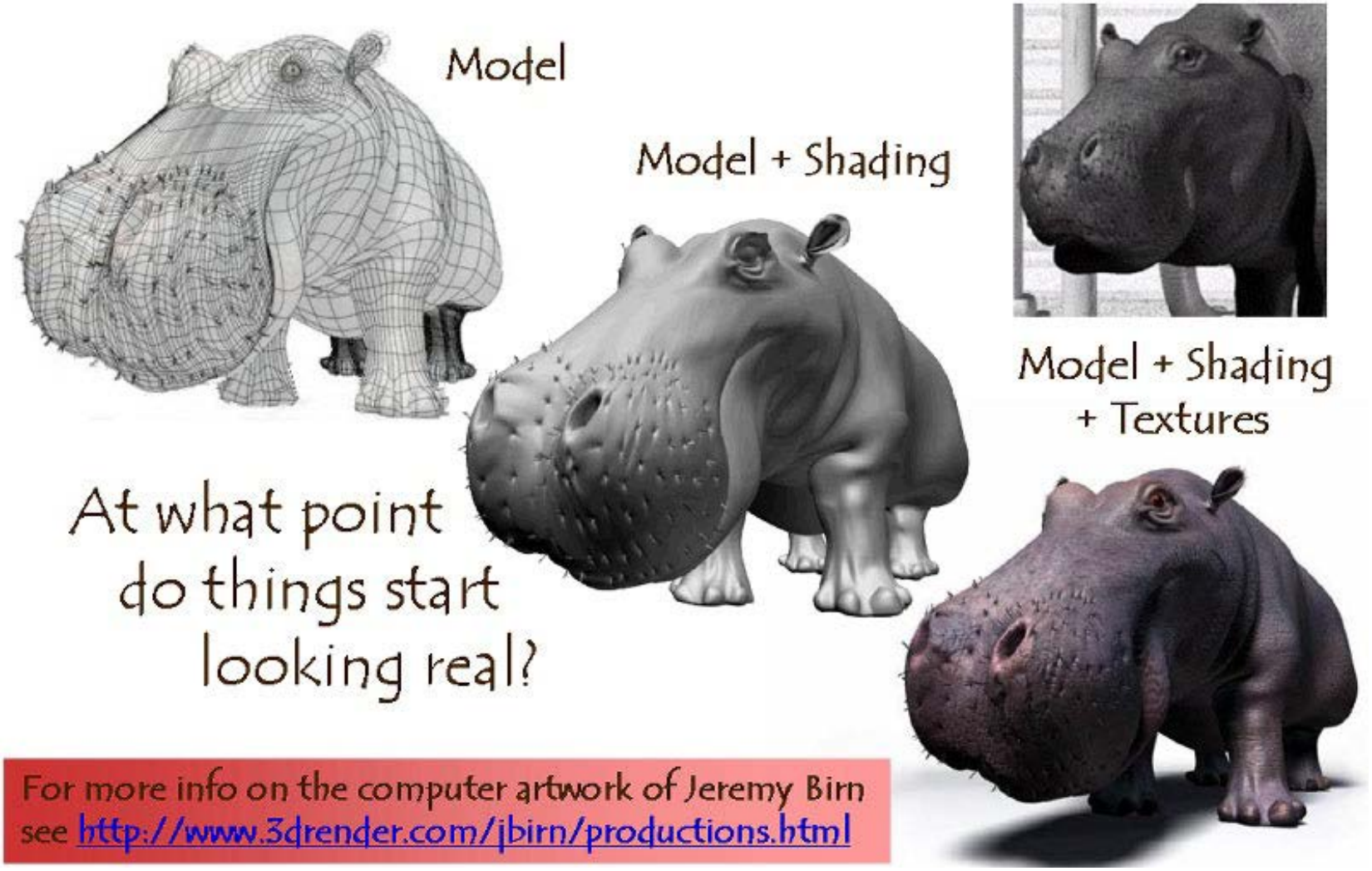
CSC 4356
Interactive Computer Graphics
Lecture 18: Texture Mapping (part 1)

Jinwei Ye

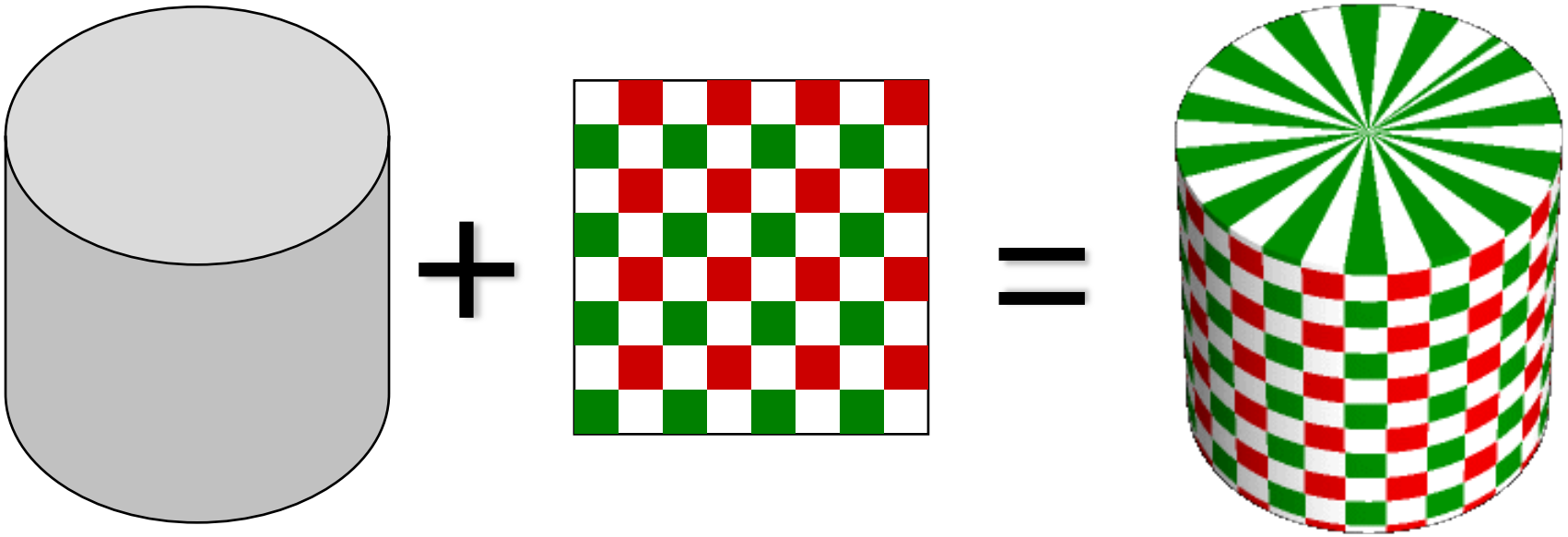
http://www.csc.lsu.edu/~jye/CSC4356/

Tue & Thu: 10:30 - 11:50am
218 Tureaud Hall

The Quest for Visual Realism



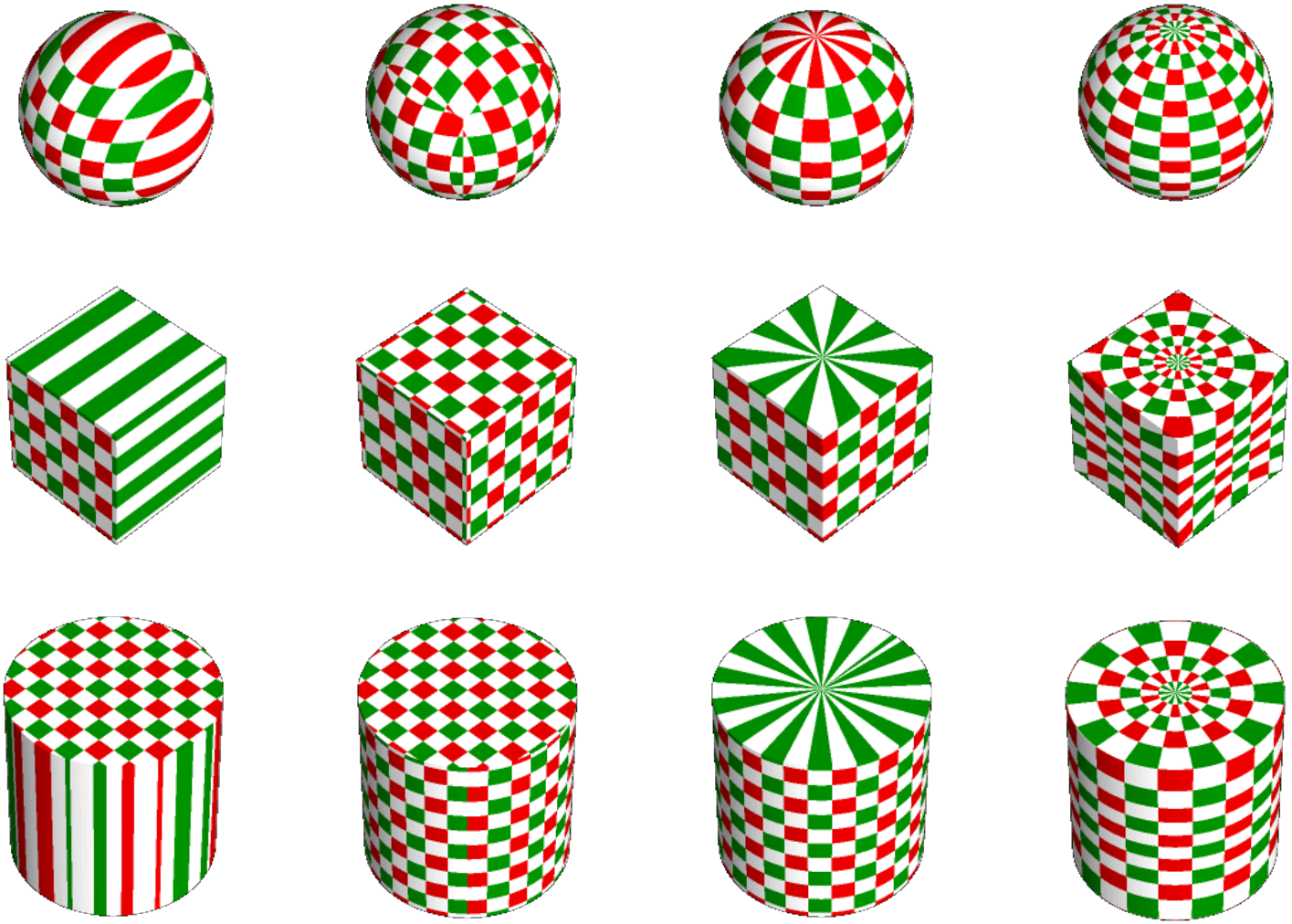
Parameterization



geometry **image** **texture map**

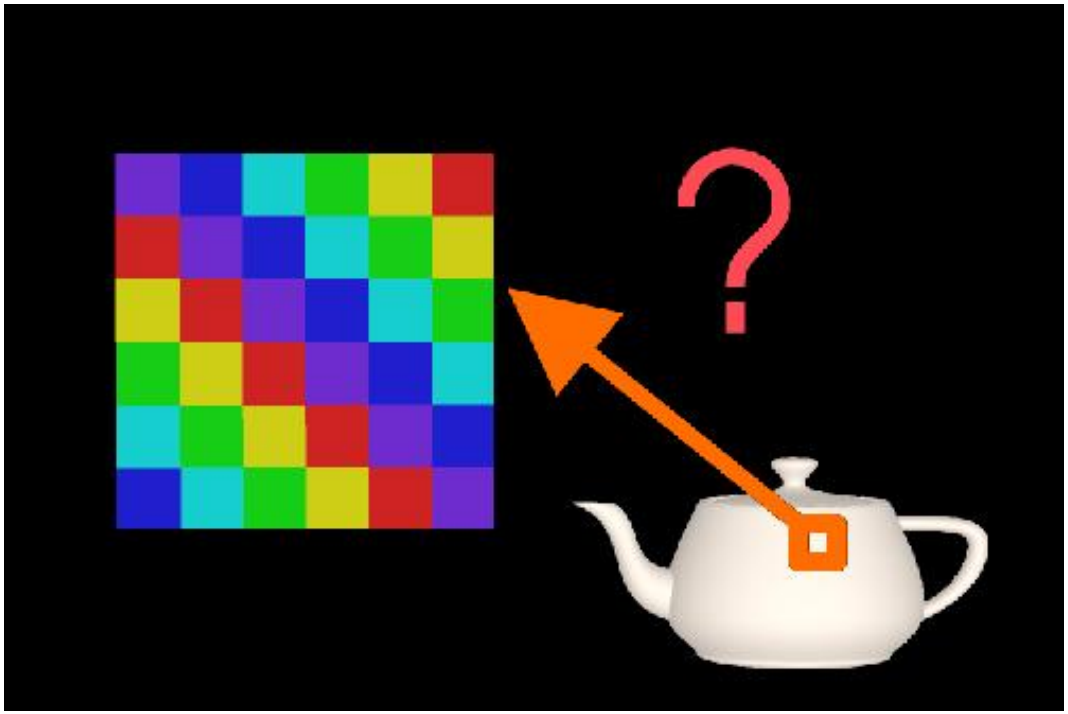
- Q: How do we decide *where* on the geometry each color from the image should go?

Options: Varieties of Mappings



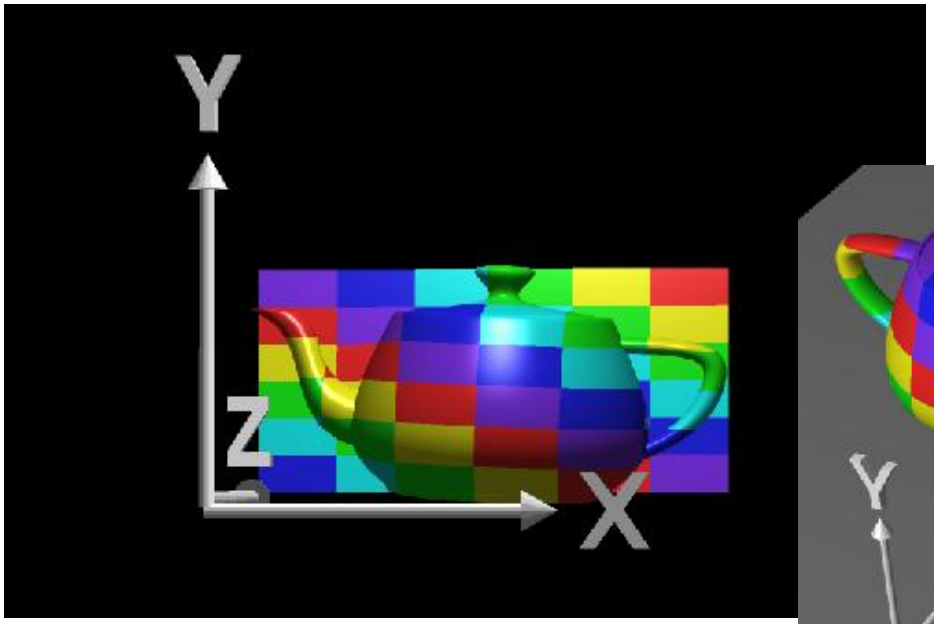
How to map object to texture?

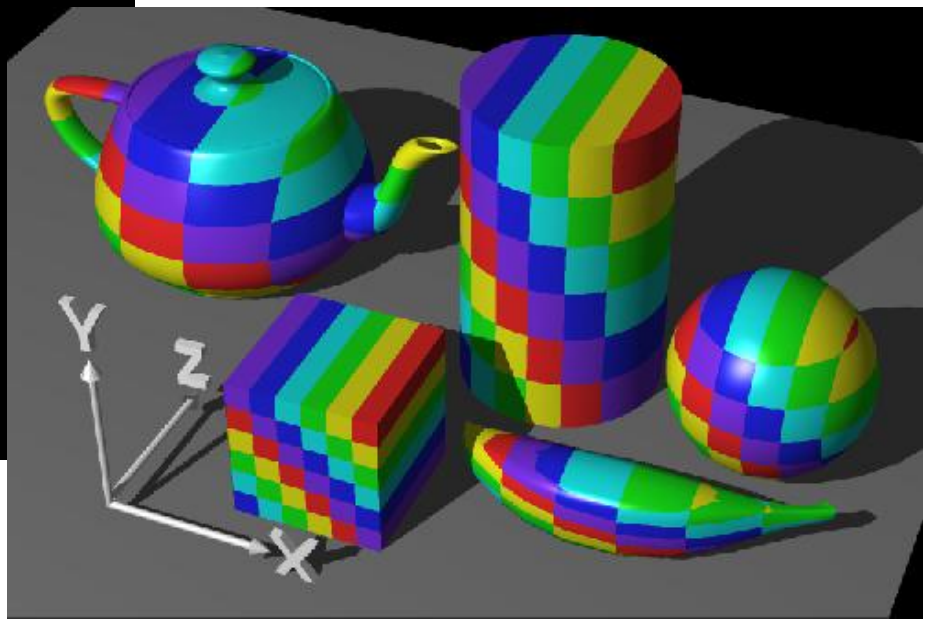
- To each vertex (x,y,z in object coordinates), must associate 2D texture coordinates (s,t)
- So texture fits “nicely” over object



Planar Mapping

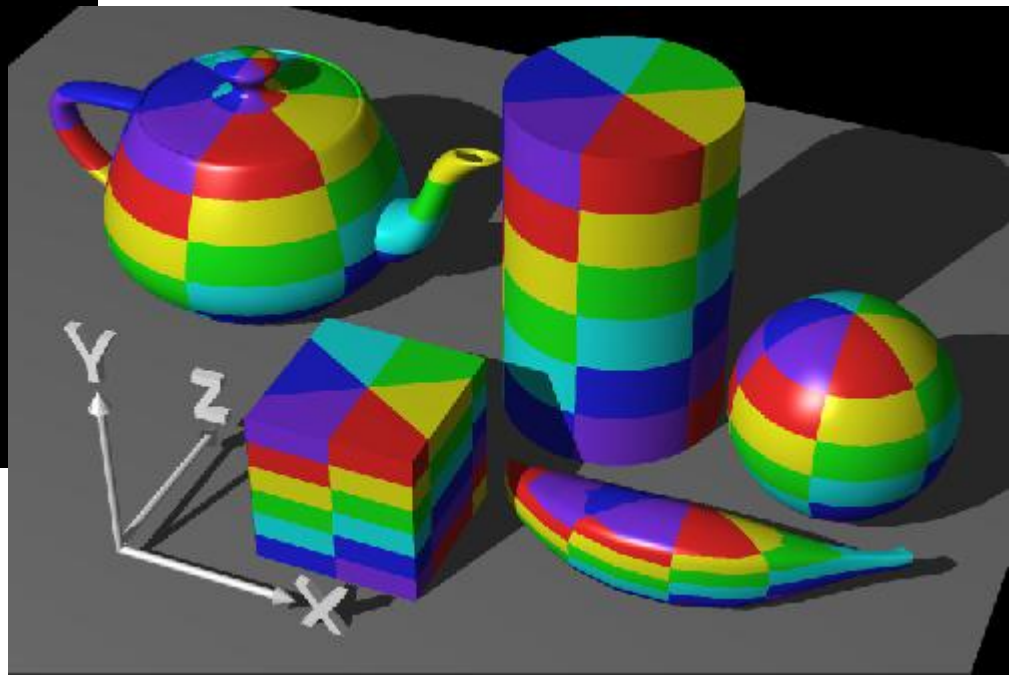
- Like projections, take vertex coordinate (x,y,z) and throw away one dimention
  - e.g., drop z such that texture coord (u,v) = (x/W,y/H)





Cylindrical Mapping

- Cylinder: r, θ, z with (u,v) = (θ/(2π),z)
  - Note seams when wrapping around (θ = 0 or 2π)

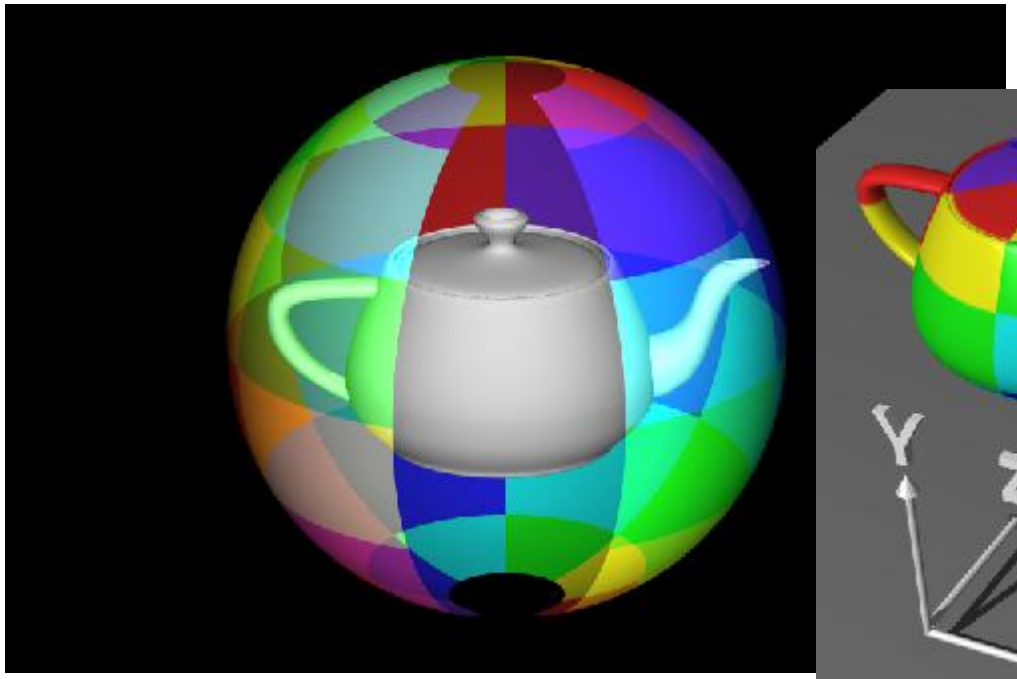


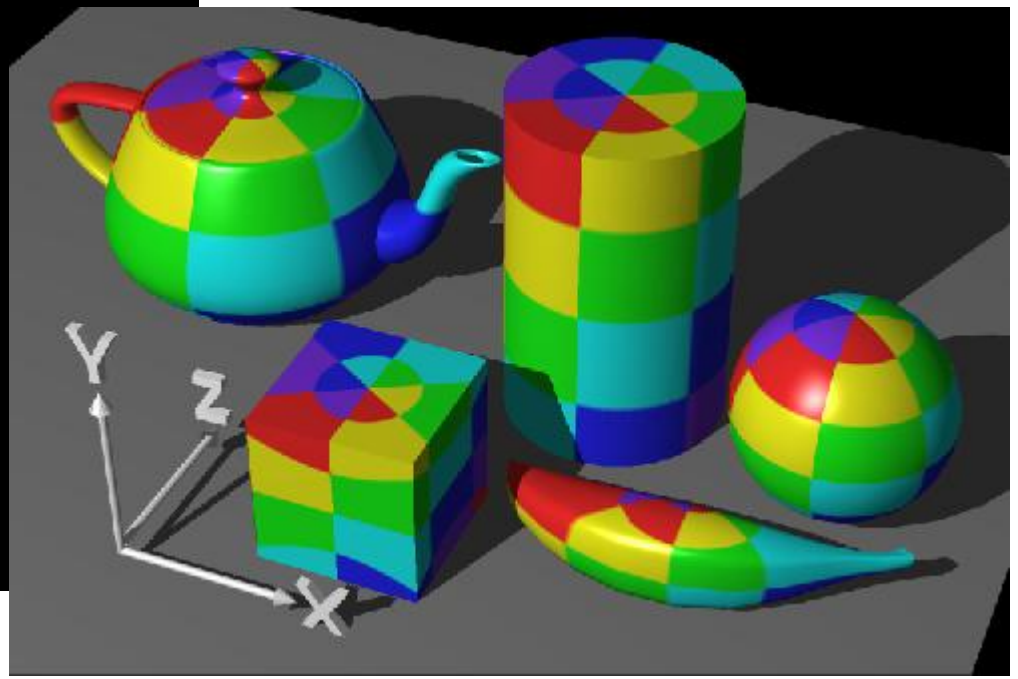
Basic Mapping Procedure

- First, map (square) texture to basic map shape
- Then, map basic map shape to object
  - Or vice versa: Object to map shape, map shape to square
- Usually, this is straightforward
  - Maps from square to cylinder, plane, …
  - Maps from object to these are simply coordinate transform

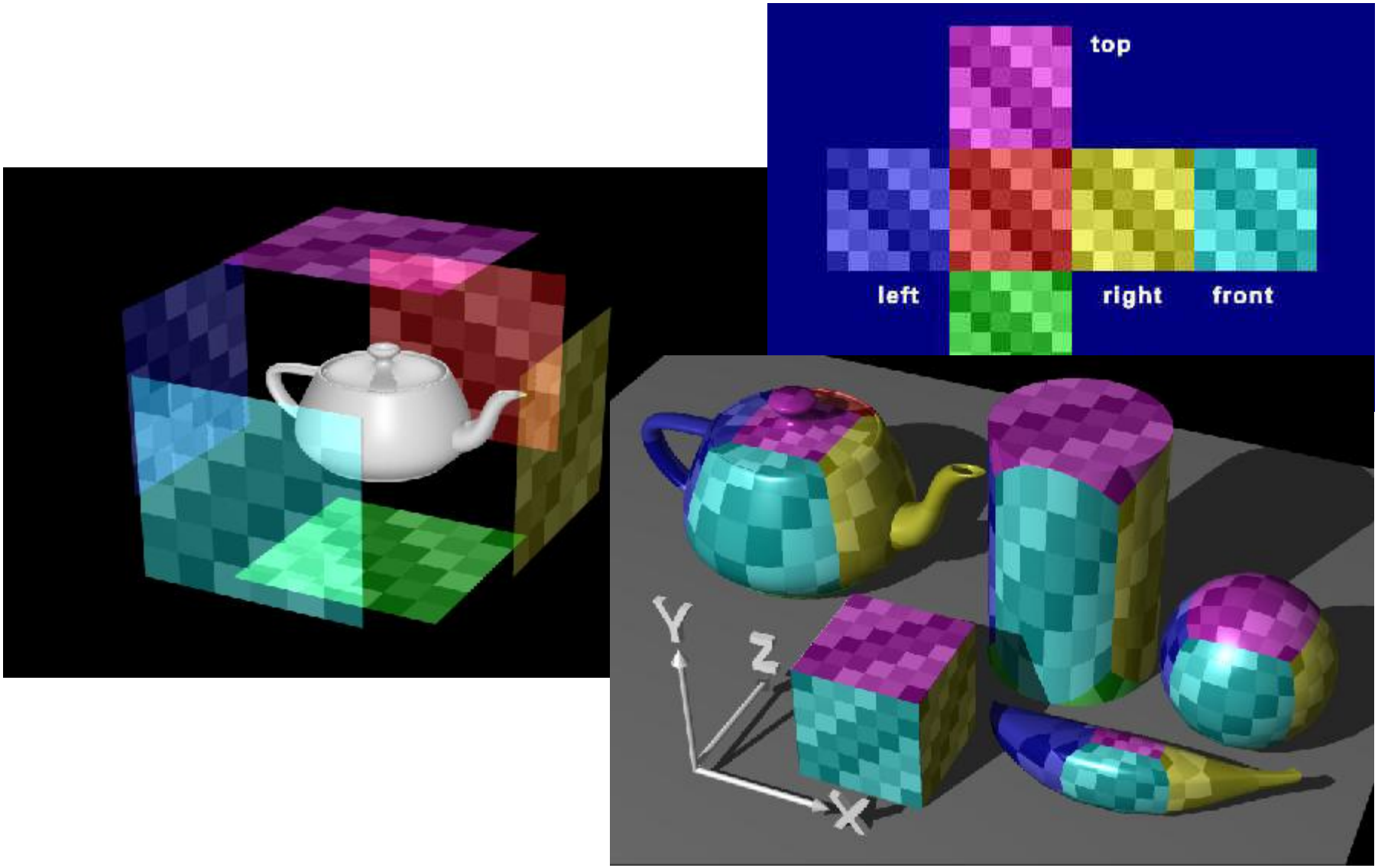
Spherical Mapping

- Convert to spherical coordinates: use latitude/longitude
  - Singularities at north and south poles



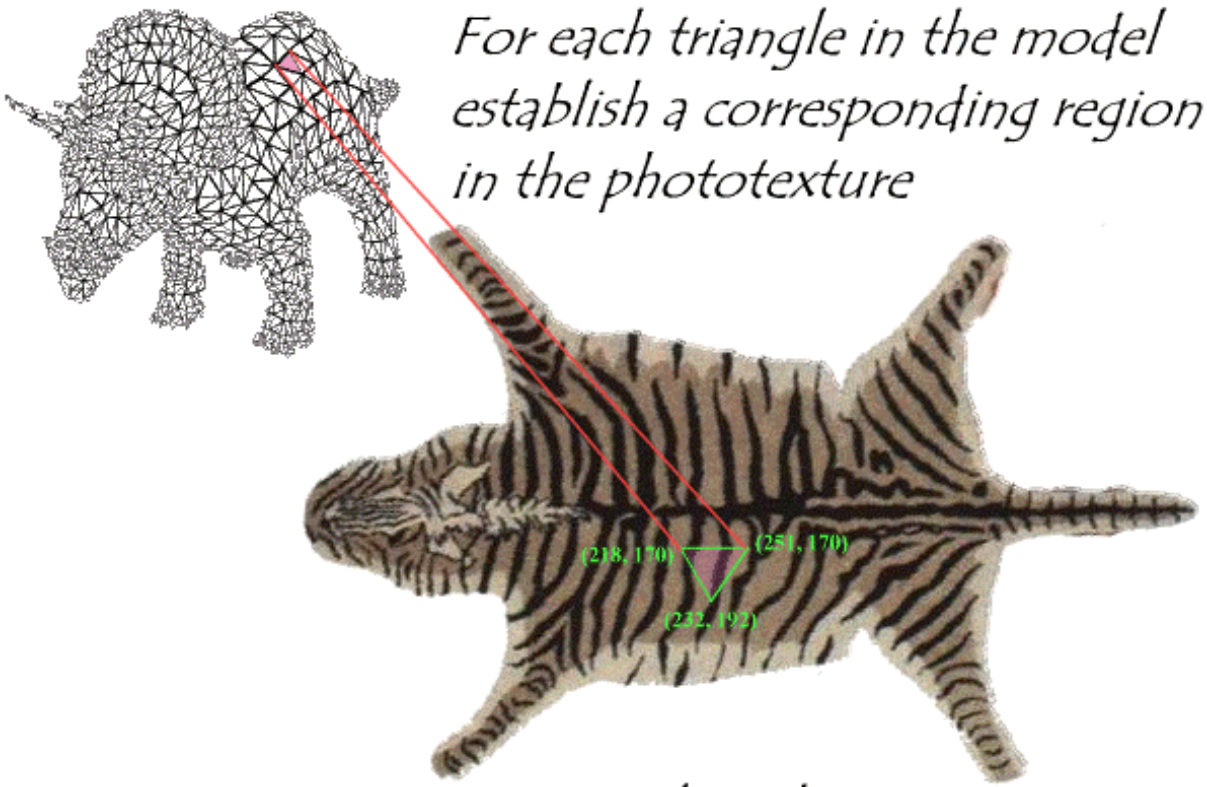


Cube Mapping

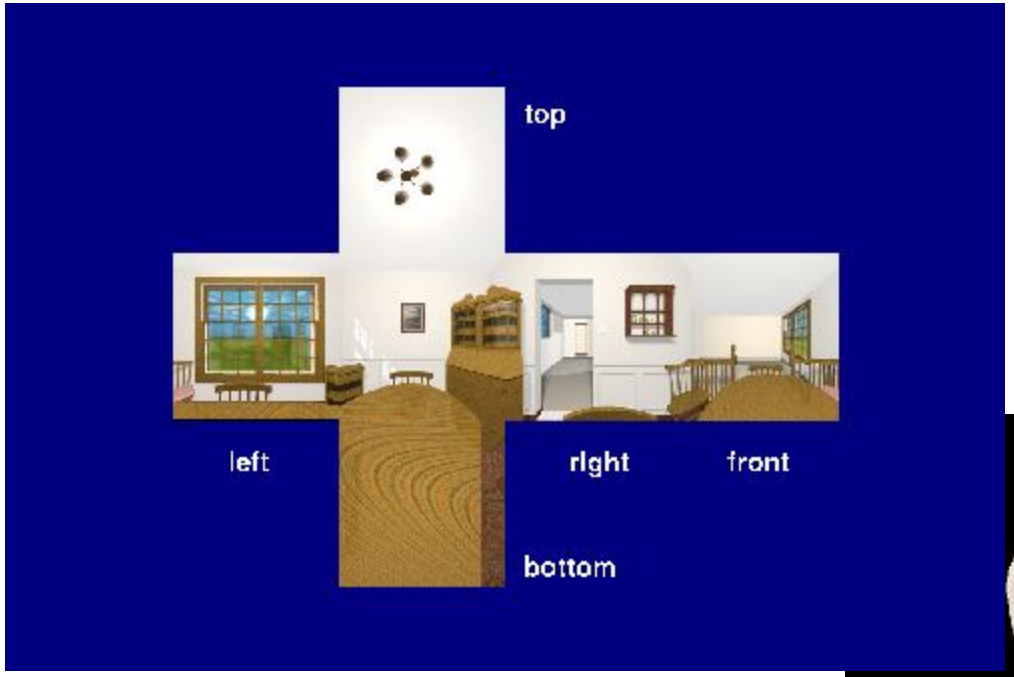


Decal Textures

- The concept is very simple



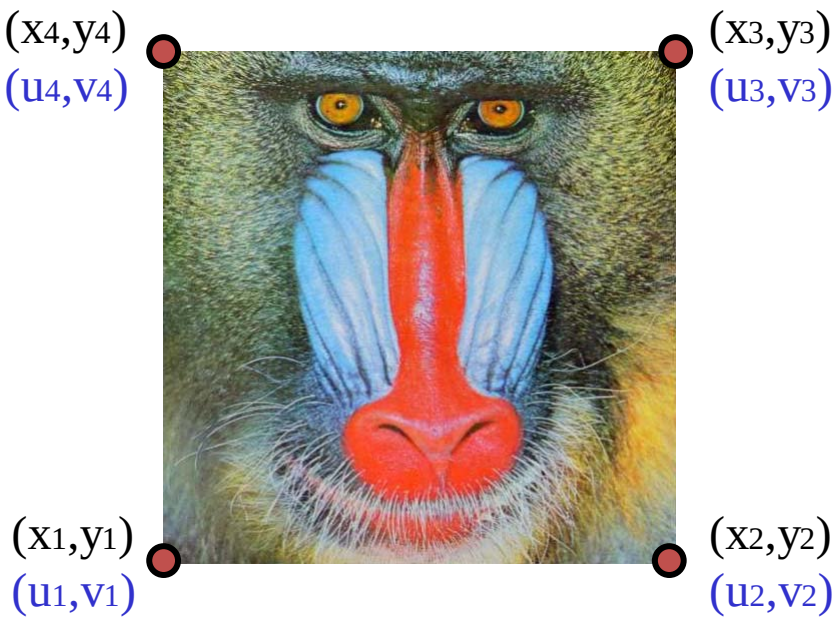
Questions?



Texture maps in OpenGL

- Specify normalized texture coordinates at each of the vertices (u, v)
  - Within range [0,1]
- Texel indices

  (s,t) = (u, v) ·(width, height)
- Texture dimensions are usually a power of 2

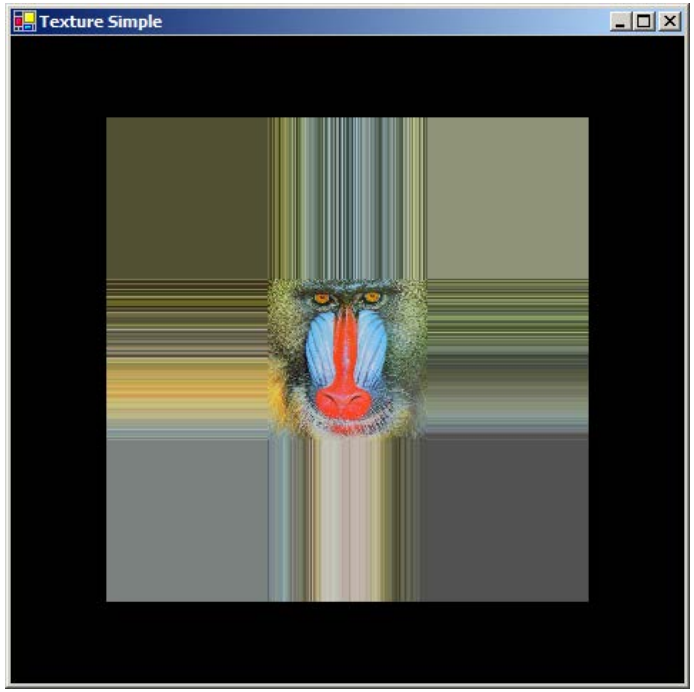



```
void Draw() {
glClear(GL_COLOR_BUFFER_BIT
| GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
:
// Draw Front of the Cube
glEnable(GL_TEXTURE_2D);
between and 1
glBegin(GL_QUADS);
glTexCoord2d(0, 1);
glVertex3d( 1.0, 1.0, 1.0);
glTexCoord2d(1, 1);
glVertex3d(-1.0, 1.0, 1.0);
glTexCoord2d(1, 0);
glVertex3d(-1.0,-1.0, 1.0);
glTexCoord2d(0, 0);
glVertex3d( 1.0,-1.0, 1.0);
glEnd();
glDisable(GL_TEXTURE_2D);
:
glFlush();
}
```

Wrapping

- The behavior of texture coordinates outside of the range [0,1] is determined by the texture wrap options.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap_mode)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap_mode)
```

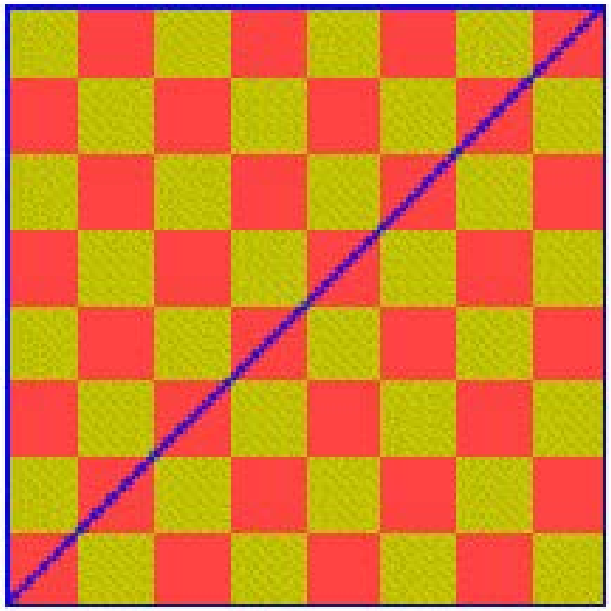


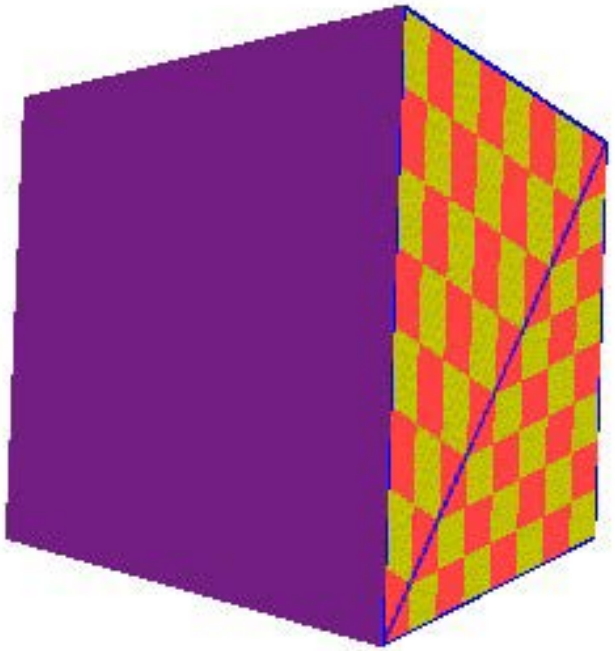
GL_CLAMP

GL_REPEAT

Linear Interpolation of Texture Coordinates

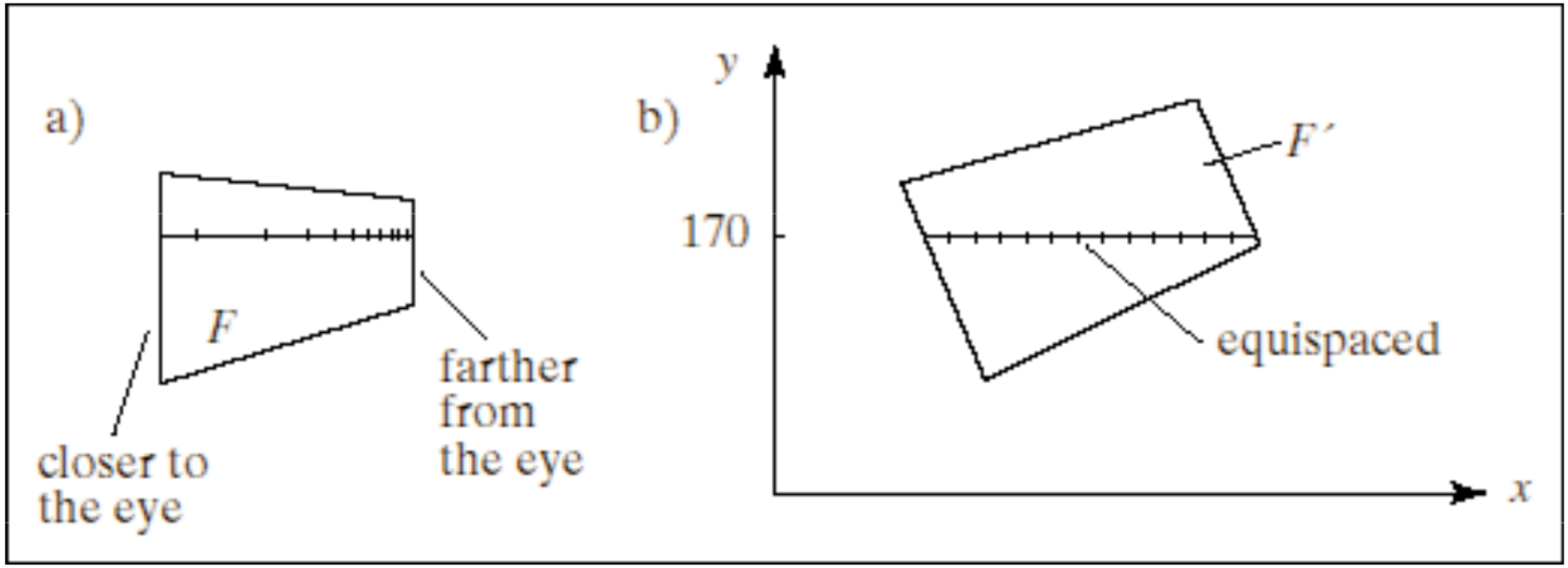
- Simple linear interpolation of u and v over a triangle leads to unexpected results
  - Same linear interpolation as used in z-buffer, Gouraud shading
  - Distortion when the triangle vertices don't have the same depth
  - Noticable during animation



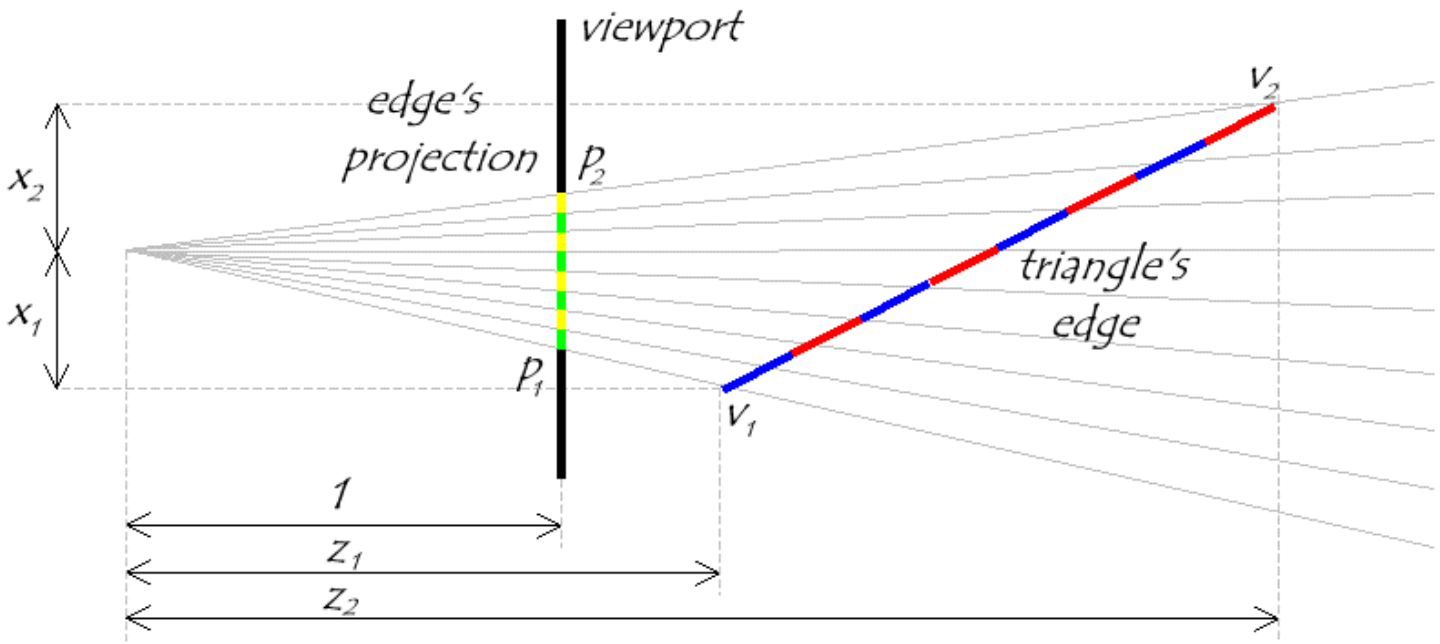


Why?

- Equal spacing in screen (pixel) space is NOT the same as in eye (texture) space in perspective projection
  - Perspective foreshortening

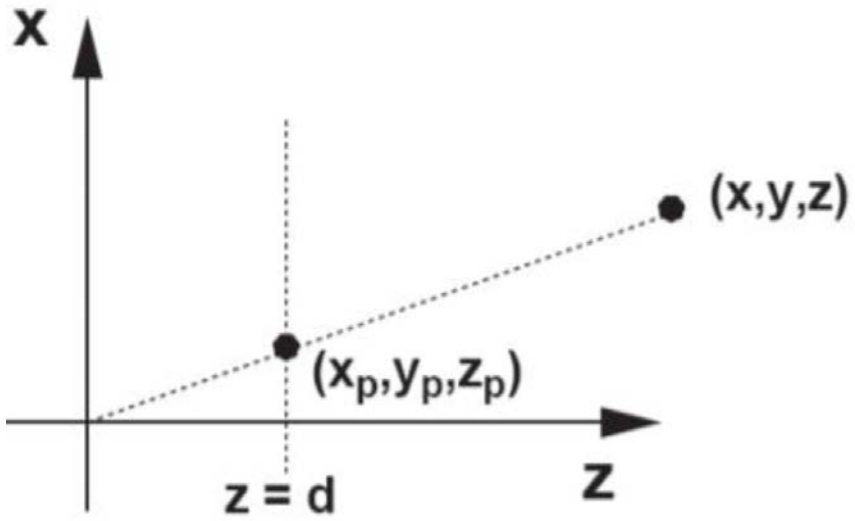


Linear Interpolation of Texture Coordinates



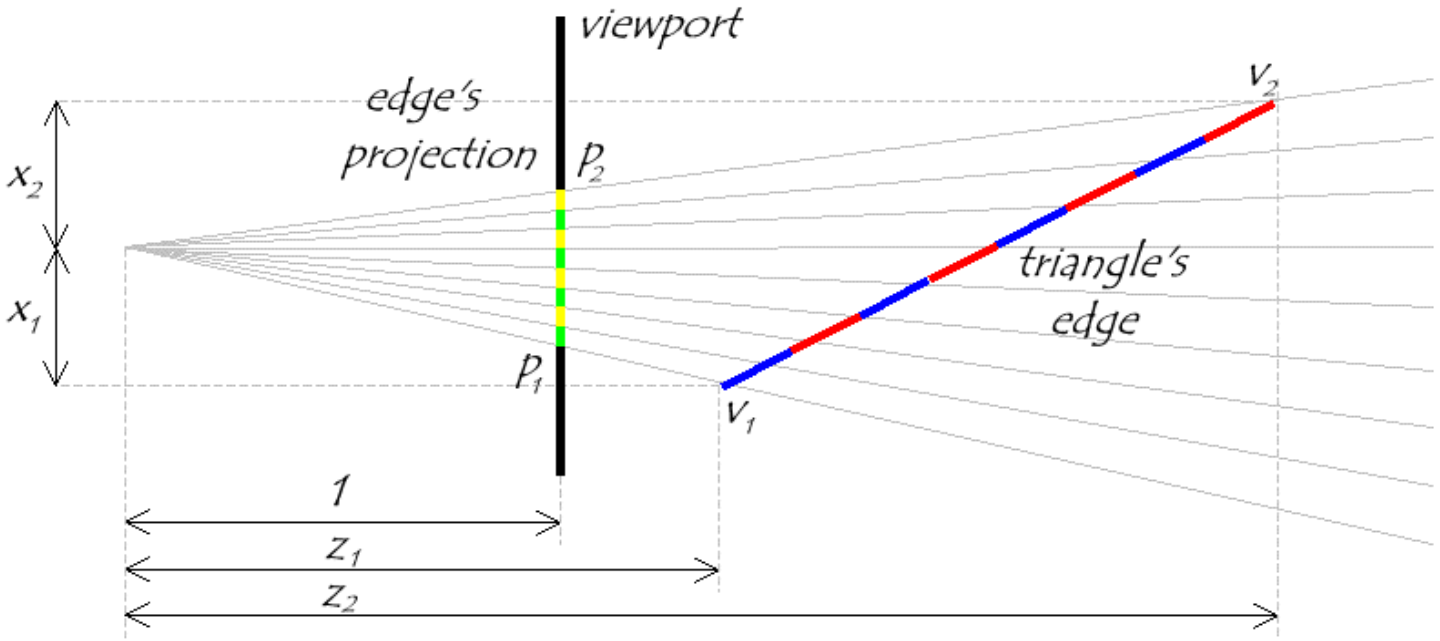
- Uniform steps along the edge projection in screen space do not correspond to uniform steps along the actual edge in eye space

Perspective Projection



$$\begin{bmatrix} wx' \\ wy' \\ wz' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

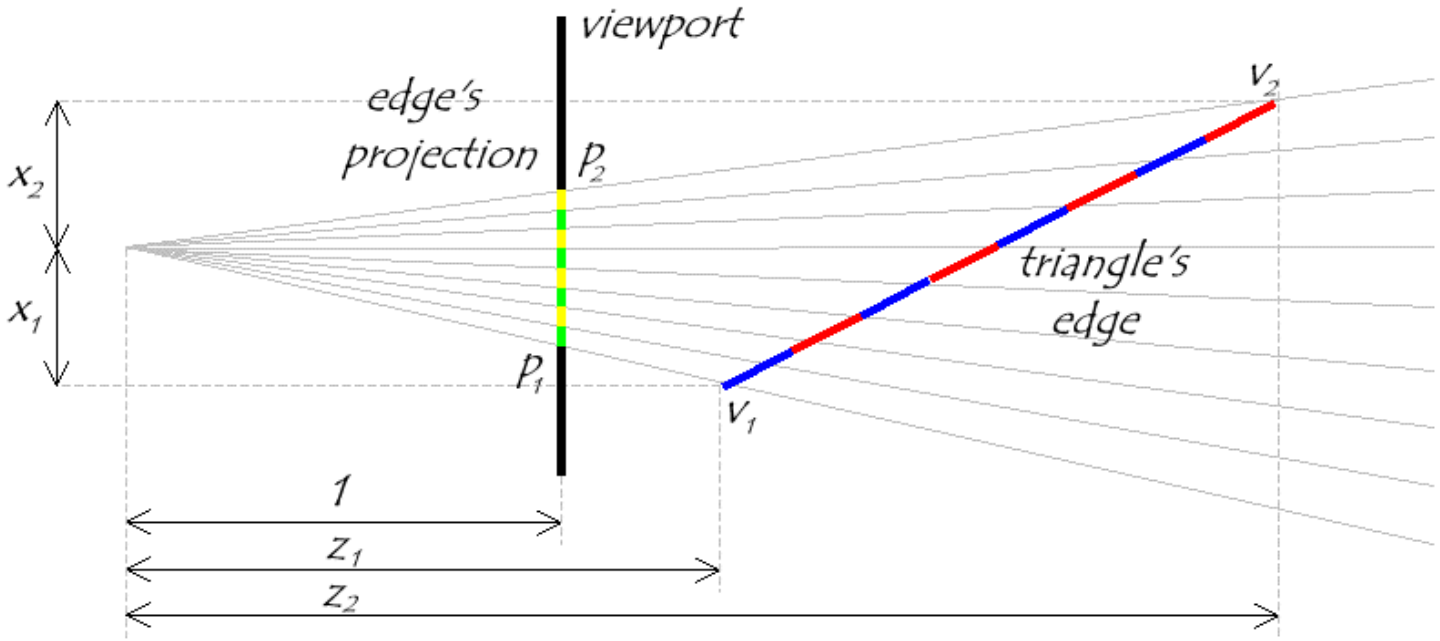
Linear Interpolation in Screen Space



Linear interpolation in screen space

$$p(t) = p_1 + t(p_2 - p_1) = \frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right)$$

Linear Interpolation in Eye Space



Linear interpolation in eye space:

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + s\left(\begin{bmatrix} x_2 \\ z_2 \end{bmatrix} - \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} \right)$$

$$P\left(\begin{bmatrix} x \\ z \end{bmatrix} \right) = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

Projection in screen space after interpolation

Correcting the Interpolation

We want to interpolate in eye space, but in terms of our screen space. So we need to find a mapping from *t* values to *s* values.

$$\frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right) = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

Solve for *s* in terms of *t* giving:

$$s = \frac{t\ z_1}{z_2 + t(z_2 - z_1)}$$

Unfortunately, at this point in the pipeline (after projection) we no longer have z_1 and z_2 lingering around (Why?). However, we do have $w_1 = 1/z_1$ and $w_2 = 1/z_2$.

$$s = \frac{t\ \frac{1}{w_1}}{\frac{1}{w_2} + t(\frac{1}{w_2} - \frac{1}{w_1})} = \frac{t\ w_2}{w_1 + t(w_2 - w_1)}$$

Interpolating Parameters

We can now use this expression for s to interpolate arbitrary parameters, such as texture indices *(u, v),* over our 3-space triangle. This is accomplished by substituting our solution for *s* given *t* into the parameter interpolation

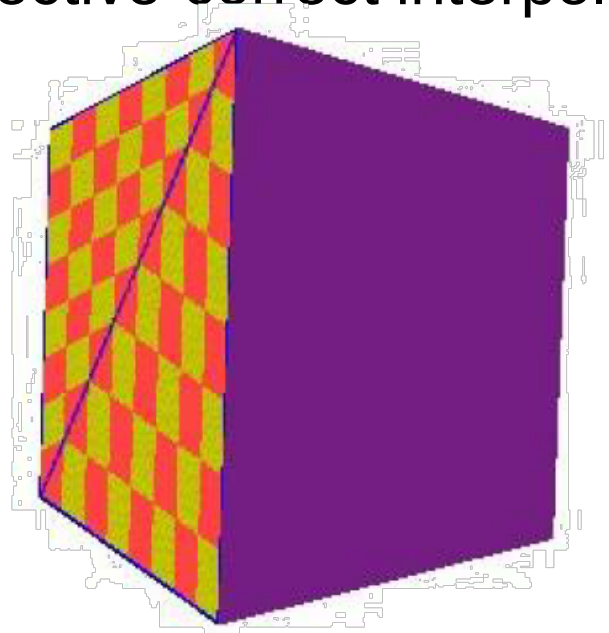
$$u = u_2 + s(u_2 - u_1)$$

$$u = u_2 + \frac{t\ w_2}{w_1 + t(w_2 - w_1)}(u_2 - u_1) = \frac{u_1 w_1 + t\ (u_2 w_2 - u_1 w_1)}{w_1 + t(w_2 - w_1)}$$

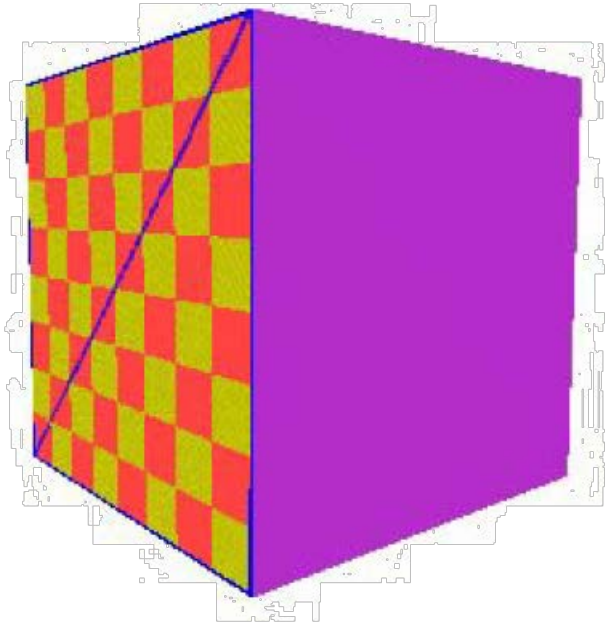
Therefore, if we **pre-multiply all parameters that we wish to interpolate in eye space by their corresponding *w* value** and add a new plane equation to interpolate the *w* values themselves, we can interpolate the numerators and denominator in screen-space. We then need to perform a divide to map the screen-space interpolants to their corresponding eye space values.

Perspective-Correct Interpolation

For obvious reasons this method of interpolation is called ***perspective-correct*** interpolation. The fact is, the name could be shortened to simply correct interpolation. You should be aware that not all 3-D graphics APIs implement perspective-correct interpolation



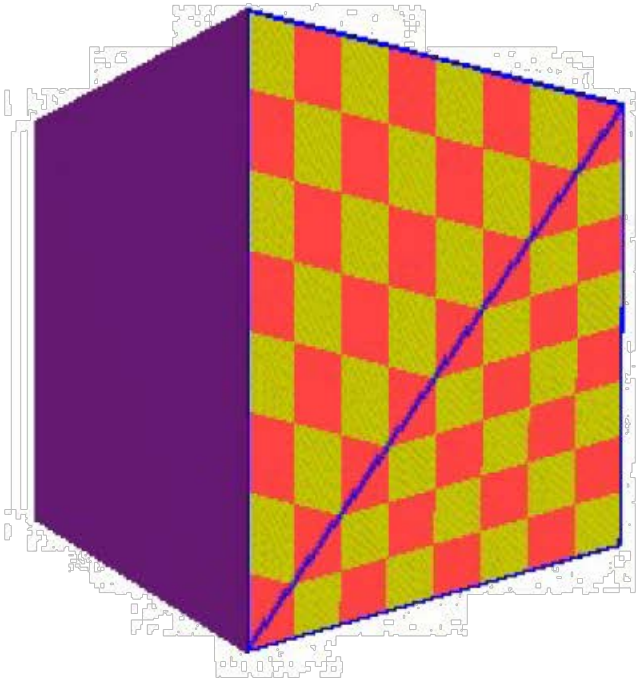
linear interpolation

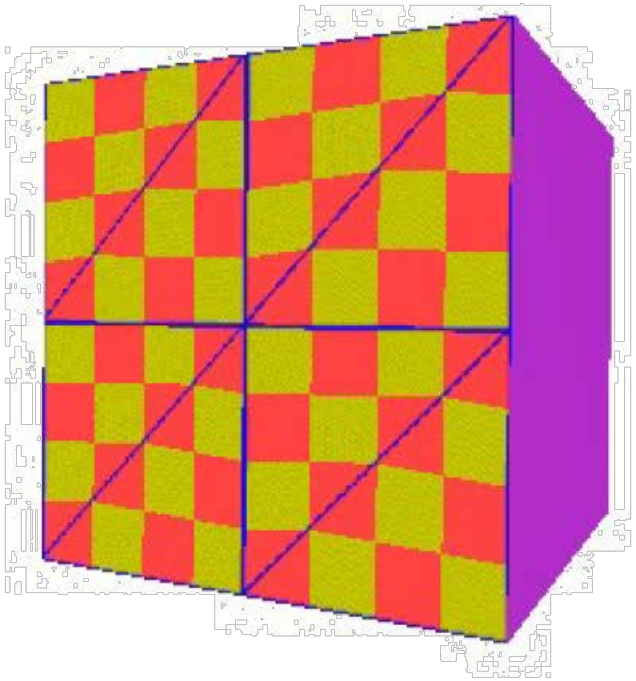


perspective-correct interpolation

Dealing with Incorrect Interpolation

- You can reduce the perceived artifacts of non-perspective correct interpolation by subdividing the texture-mapped triangles into smaller triangles (why does this work?). But, fundamentally the screen-space interpolation of projected parameters is wrong





Perspective Correction Hint

- Texture coordinate and color interpolation:
  - Linearly in screen space (wrong) **OR**
  - Perspective correct interpolation (slower)
- glHint (GL_PERSPECTIVE_CORRECTION_HINT, **hint**); where **hint** is one of:
  - GL_NICEST: Perspective
  - GL_FASTEST: Linear
  - GL_DONT_CARE: Linear

Review of Textures

- Increases the apparent complexity of simple geometry
- Requires perspective projection correction
- Specifies variations in shading within a primitive:
  - Surface Reflectance