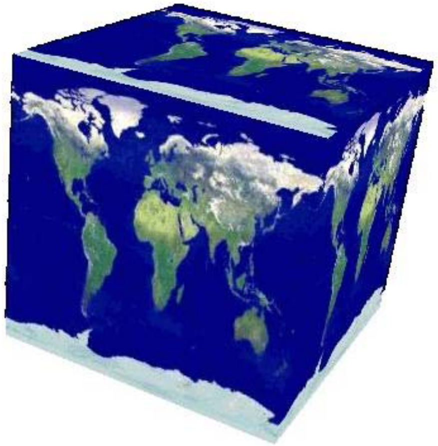
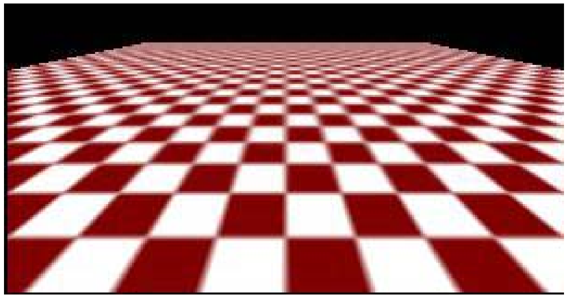
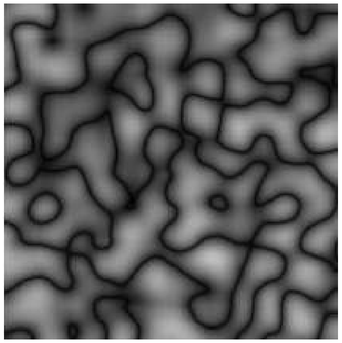
Texture Mapping



- A way of adding surface details
- Two ways can achieve the goal:
  - Surface detail polygons: create extra polygons to model object details
    - Add scene complexity and thus slow down the graphics rendering speed
    - Some fine features are hard to model!
  - ✓ Map a texture to the surface (a more popular approach)

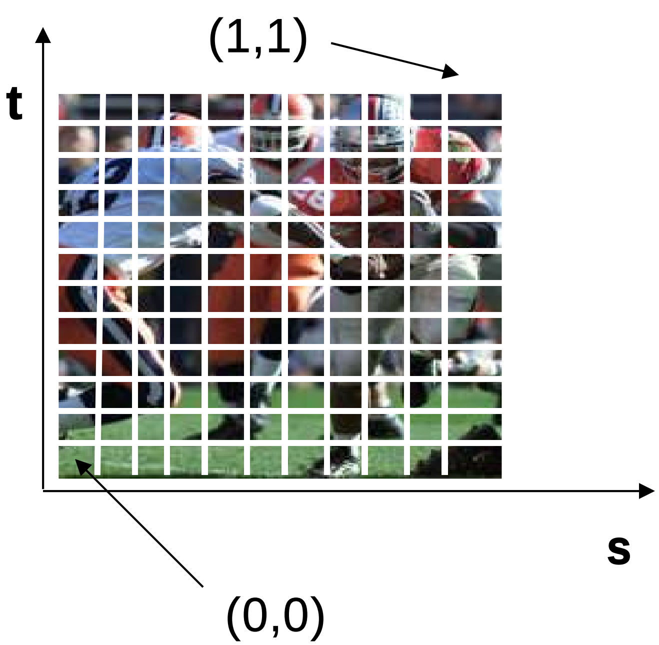




Complexity of images does
Not affect the complexity
Of geometry processing
(transformation, clipping...)

Texture Representation

- ✓ Bitmap (pixel map) textures (supported by OpenGL)
- Procedural textures (used in advanced rendering programs)



Bitmap texture:

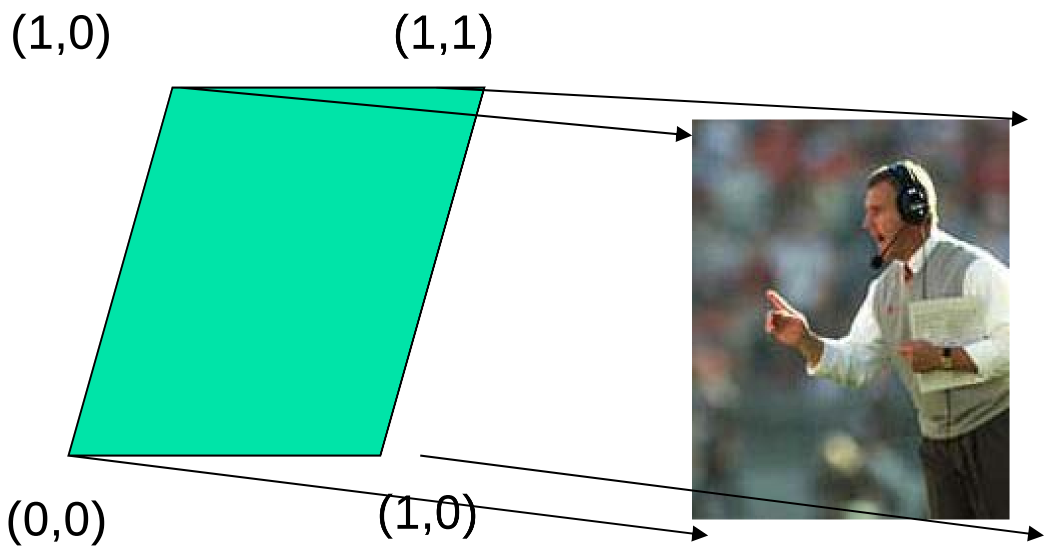
- A 2D image - represented by 2D array texture[height][width]
- Each pixel (or called **texel**) by a unique pair texture coordinate (s, t)
- The s and t are usually normalized to a [0,1] range
- For any given (s,t) in the normalized range, there is a unique image value (i.e., a unique [red, green, blue] set)

Map textures to surfaces

- Establish mapping from texture to surfaces (polygons):

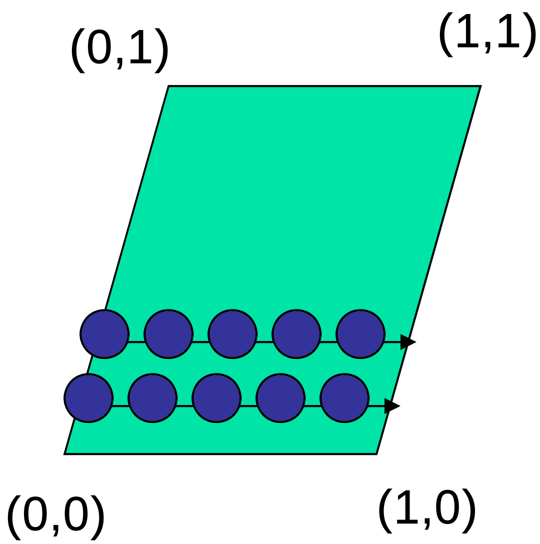
  - Application program needs to specify texture coordinates for each corner of the polygon

The polygon can be in an arbitrary size



Map textures to surfaces

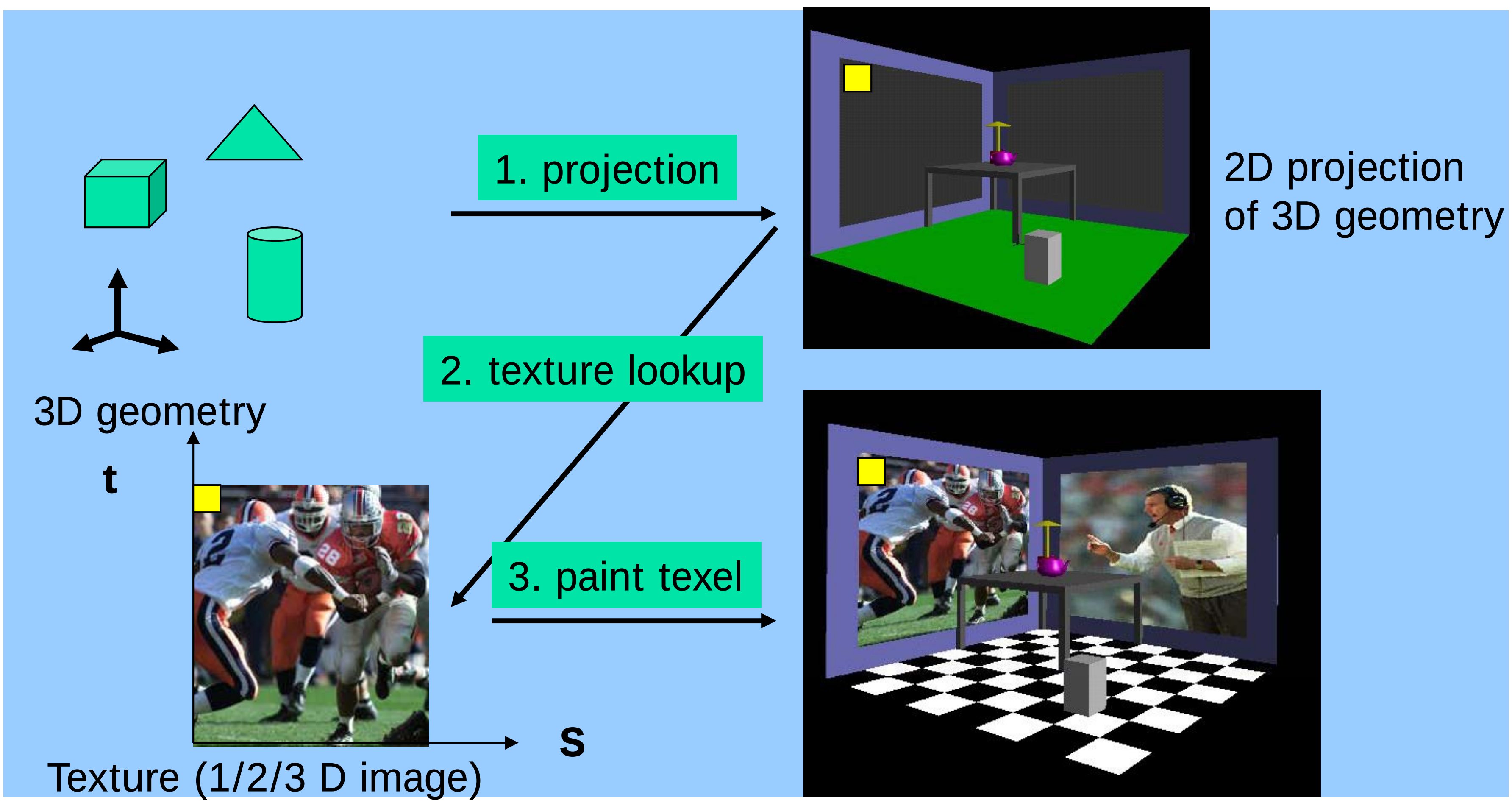
- Texture mapping is performed in rasterization (backward mapping)



❑ For each pixel that is to be painted, its texture coordinates (s, t) are determined (interpolated) based on the corners' texture coordinates (why not just interpolate the color?)

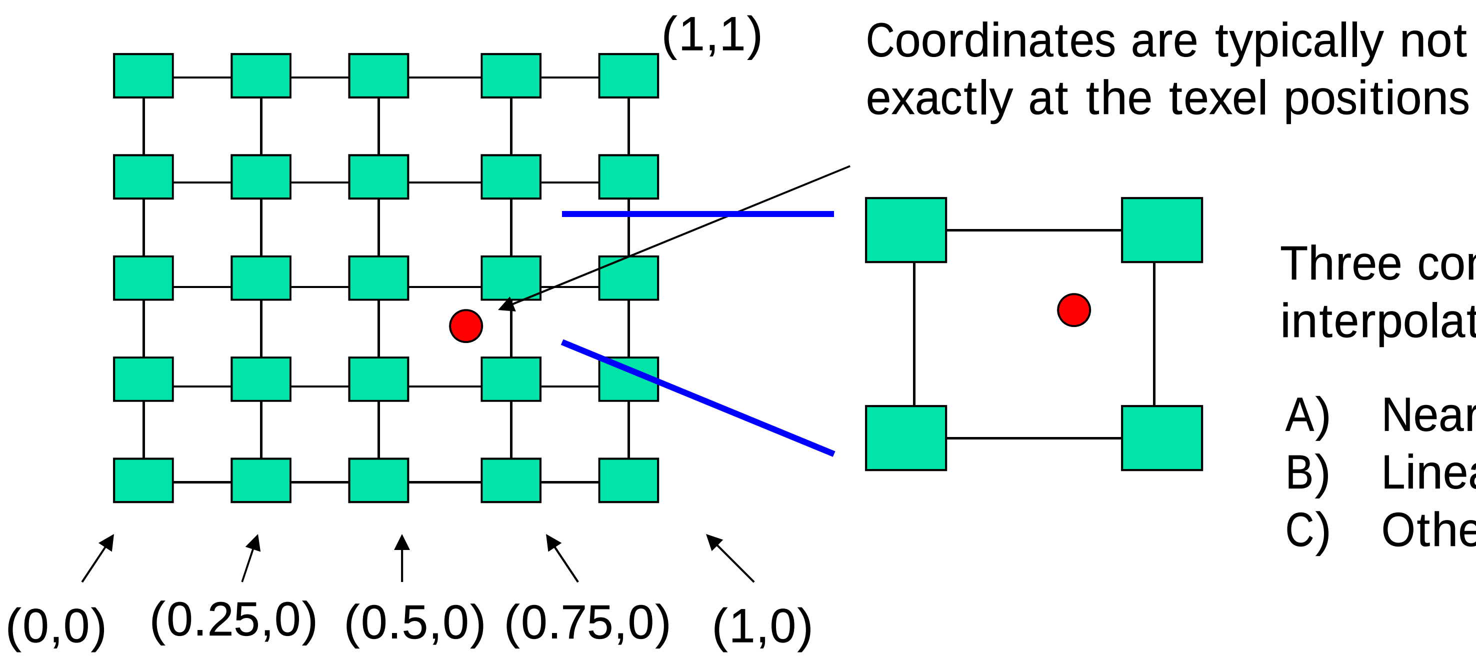
❑ The interpolated texture coordinates are then used to perform texture lookup

Texture Mapping



Texture Value Lookup

- Texture coordinates, like other vertex attributes, are interpolated in screen space
- For the given texture coordinates (s,t), we can find a unique image value from the texture map



Three common interpolation methods:

A) Nearest neighbor
B) Linear Interpolation
C) Other filters

OpenGL texture mapping setup

- Steps in your program

1) Specify texture
   - read or generate images
   - generate texture objects (optional)
   - Assign images to textures
2) Specify texture mapping parameters
   - Wrapping, filtering, etc.
3) Enable GL texture mapping, e.g. GL_TEXTURE_2D
4) Assign texture coordinates to vertices
5) Draw your objects (with fixed function pipeline or shaders)
6) Disable GL texture mapping (if you don't need to perform texture mapping any more)

Specify textures

- Load the texture map from main memory to graphics card (texture memory)
  - glTexImage2D(Glenum target, Glint level, Glint iformat, int width, int height, int border, Glenum format, Glenum type, Glvoid* img)
- Example:
  - glTeximage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB, GL_UNSIGNED_BYTE, myImage);

    (myImage is a 2D array: GLuByte myImage[64][64][3];)
- The dimensions of texture images usually are powers of 2; but OpenGL also supports non power of 2 (GL_TEXTURE_RECTANGLE)

Fix texture size

- If the dimensions of the texture map are not power of 2, you can

1) Pad zeros 2) Scale your image in advance

60

100

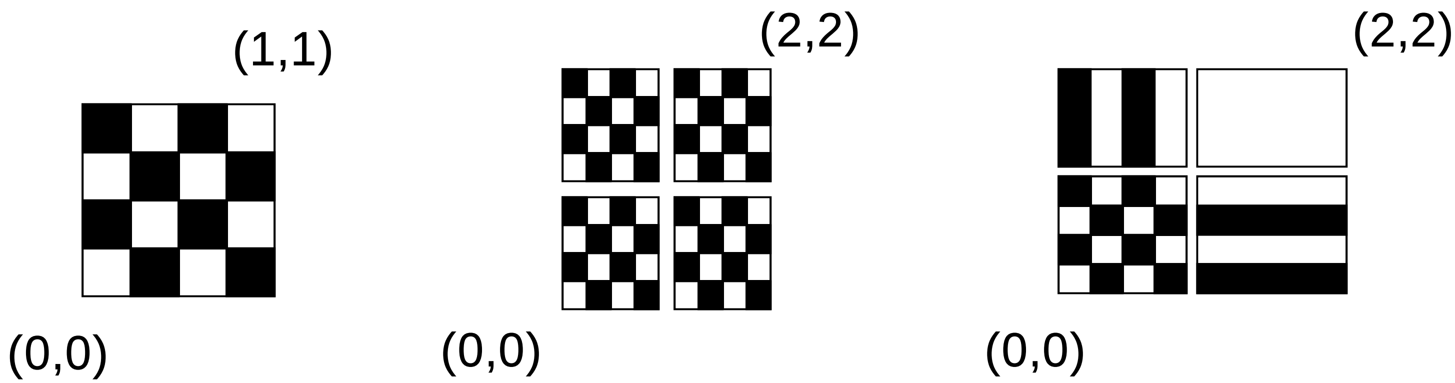
128

64

Remember to adjust the texture coordinates for your polygon corners – you don't want to Include black texels in your final picture

Texture mapping parameters

- What happen if the given texture coordinates (s,t) are outside [0,1] range?



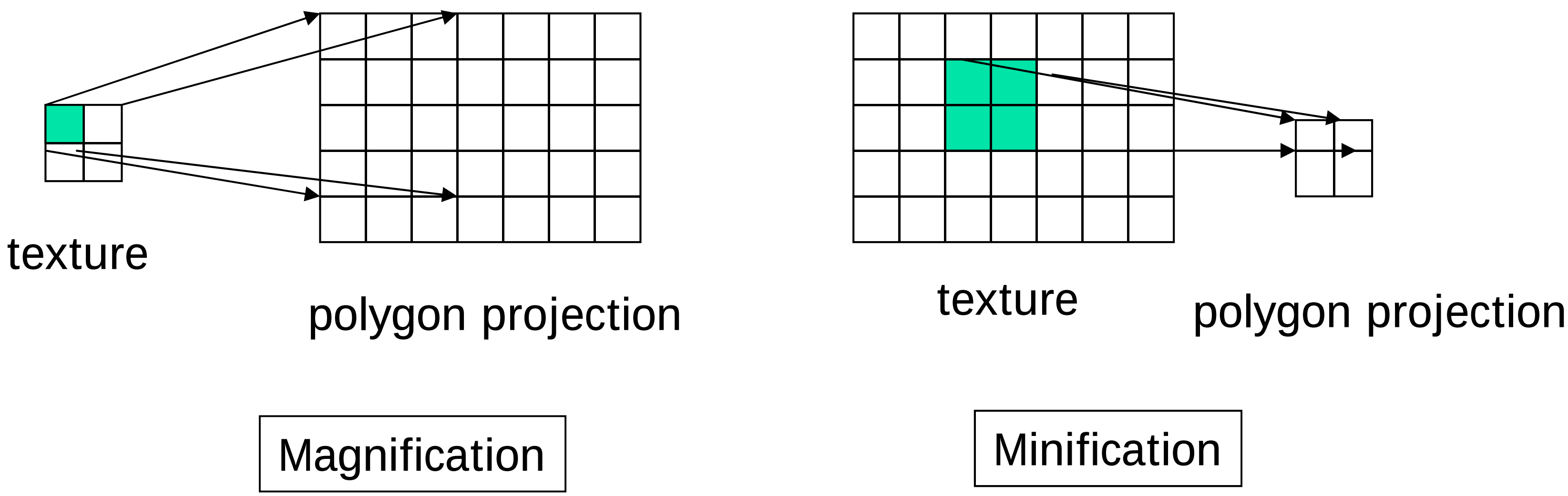
texture　　GL_Repeat　　GL_Clamp

If (s >1) s = 1
If (t >1) t = 1

- Example: glTexParameteri(GL_TEXTAURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)

Texture mapping parameters(2)

- Since a polygon can get transformed to arbitrary screen size, texels in the texture map can get magnified or minified.

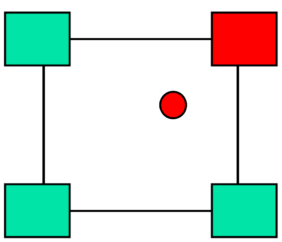


- Filtering: interpolate a texel value from its neighbors or combine multiple texel values into a single one

Texture mapping parameters(3)

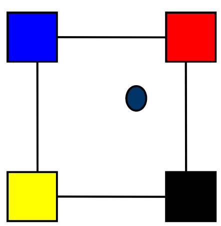
- OpenGL texture filtering:

1) Nearest Neighbor (lower image quality)



glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

2) Linear interpolate the neighbors (better quality, slower)



glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

Or GL_TEXTURE_MAX_FILTER

Texture color blending

- Determin how to combine the texel color and the object color
  - GL_MODULATE – multiply texture and object color
  - GL_BLEND – linear combination of texture and object color
  - GL_REPLACE – use texture color to replace object color

Example: glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

Enable (Disable) Textures

- Enable texture – glEnable(GL_TEXTURE_2D)
- Disable texture – glDisable(GL_TEXTURE_2D)

Remember to disable texture mapping when you draw non-textured polygons

Fixed Function Pipeline: Specify texture coordinates

- Give texture coordinates before defining each vertex

```
glBegin(GL_QUADS);
   glTexCoord2D(0,0);
   glVertex3f(-0.5, 0, 0.5);
  ...
 glEnd();
```

- These methods are depreciated
- You should use VBOs to pass the coordinates

Fixed Function Pipeline
Transform texture coordinates

- All the texture coordinates are multiplied by Gl_TEXTURE matrix before in use
- To transform texture coordinates, you do:
  - glMatrixMode(Gl_TEXTURE);
  - Apply regular transformation functions
  - Then you can draw the textured objects

Put it all together

```
…
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
...
glEnable(GL_TEXTURE_2D);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB,
    GL_UNSIGNED_BYTE, mytexture);

Draw_picture1(); // define texture coordinates and vertices in the function
....
```

Using OpenGL Texture Objects

- Avoid calling glTexImage2D (or 1D/3D etc) every time you draw
  - Not necessary if your texture is static since it will incur the actual memory transfer
- Instead, create a texture ID and associate the ID to the texture

```
glGenTextures(1, &tid);  // generate one texture handle
glBindTexture(GL_TEXTURE_2D, tid); // bind this handle to a 2D texture
glTexImage2D(GL_TEXTUER_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
      GL_UNSIGNED_BYTE, &my_texture);  // usage same as described before
glTexParameterf(.....) // set up the texture parameters as before

  ...
```

- At display time, call glBindTexture(GL_TEXTURE_2D, tid) again to use the texture for drawing

OpenGL textures in shaders

- You need to pass the texture coordinates for each vertex as attribute to the vertex shader, and then in turn pass to the fragment shader as varying
- You need to link your texture in OpenGL/C to the fragment shader as uniform variable
- You will use the GLSL build-in function *texture()* to perform texture lookup
- You need to properly blend the texture color with color from illumination calculation
- You can use multiple textures

Vertex shader example

- Use a single texture

```
attribute vec3 VertexPosition;
attribute vec3 VertexNormal;
attribute vec2 VertexTexCoord;

varying vec3  Position;
varying vec3 Normal;
varying vec2 TexCoord;

uniform mat4  ModelViewMatrix;
uniform mat4 NormalMatrix;
uniform mat4 ProjectionMatrix;
uniform mat4 MVP;
```

```
void main()  {

    TexCoord = VertexTexCoord;
    Normal = normalize(NormalMatrix *
                VertexNormal);
    Position = vec4(ModelViewMatrix *
            vec4(VertexPosition,1.0));
    gl_Position = MVP *
            vec4(VertexPosition,1.0);
}
```

Fragment Shader

```
varying vec3 Position;
varying vec3 Normal;
varying vec2 TexCoord;

Uniform sampler2D Tex1;
// parameters for lighting calculation
...
...
```

```
void main()  {

    vec4 texColor = texture(Tex1, TexCoord);
    // compute ambient, diffucse, and
    specular illuminations
    …
    ...

    gl_FragColor = vec4(ambient,1.0) +
    vec4(diffuse,1.0) * texColor +
    vec4(specular,1.0);
}
```

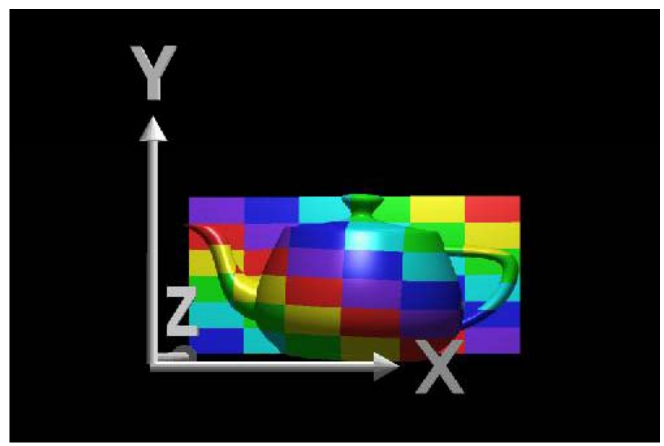
How to pass textures to your shader?

- First of all, associate your texture(s) to texture unit 0, texture unit 1 (if you have multiple textures), etc.

```
glGenTexutres(1, &tid); ┐┐┐
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tid);
glTextureImage2D(....);  // usage as described before
glTexParameterf(...) // usage as described before
int uniloc = glGetUniformLocation(program, “Tex1”);
if (uniloc >=0)  glUniformli(uniloc, 0); // associate Tex1 to texture unit 0
....
// you can create and pass multiple textures to your shader if you want
```

Projector Functions

- How do we map the texture onto a arbitrary (complex) object?
  - Construct a mapping between the 3-D point to an intermediate surface
- Idea: Project each object point to the intermediate surface with a parallel or perspective projection
  - The focal point is usually placed inside the object
  - Plane
  - Cylinder
  - Sphere
  - Cube

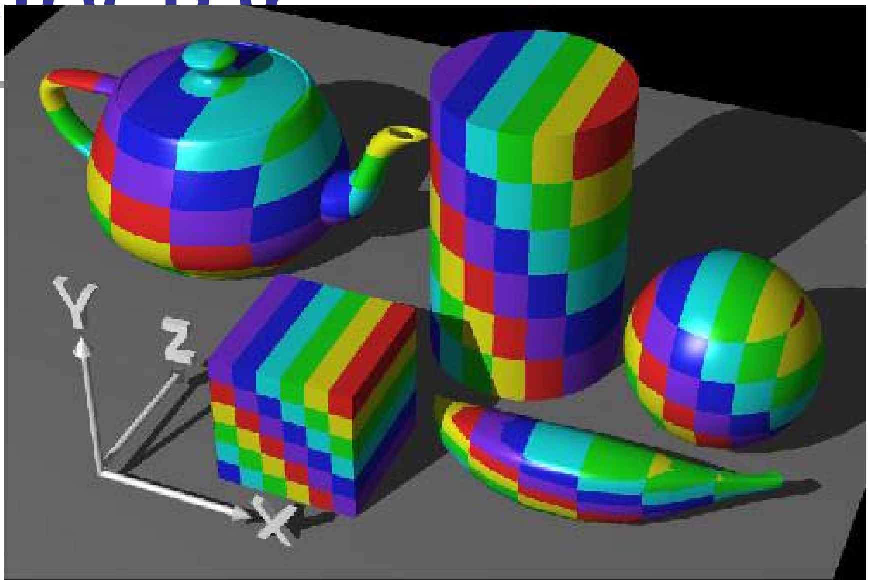


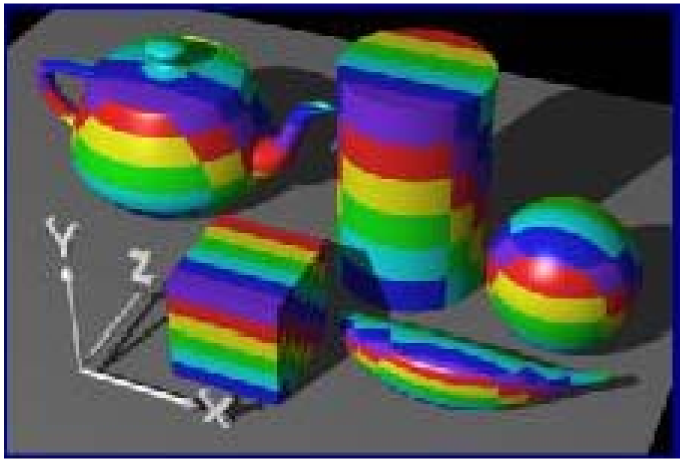
courtesy of R. Wolfe

Planar projector

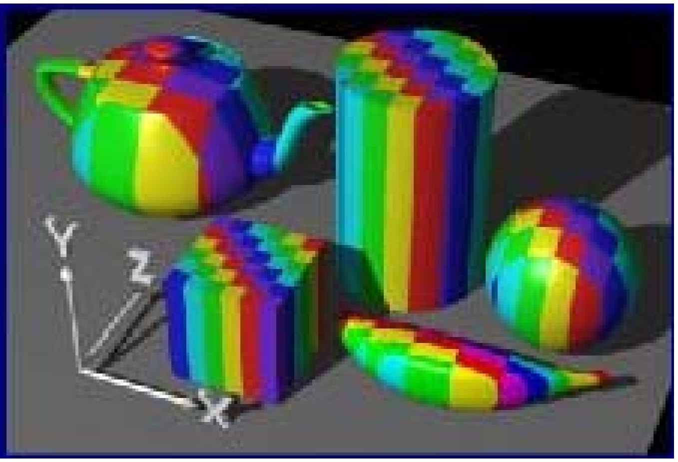
Planar Projector

Orthographic projection
onto XY plane:
$u = x, \; v = y$





...onto YZ plane



...onto XZ plane

courtesy of
R. Wolfe

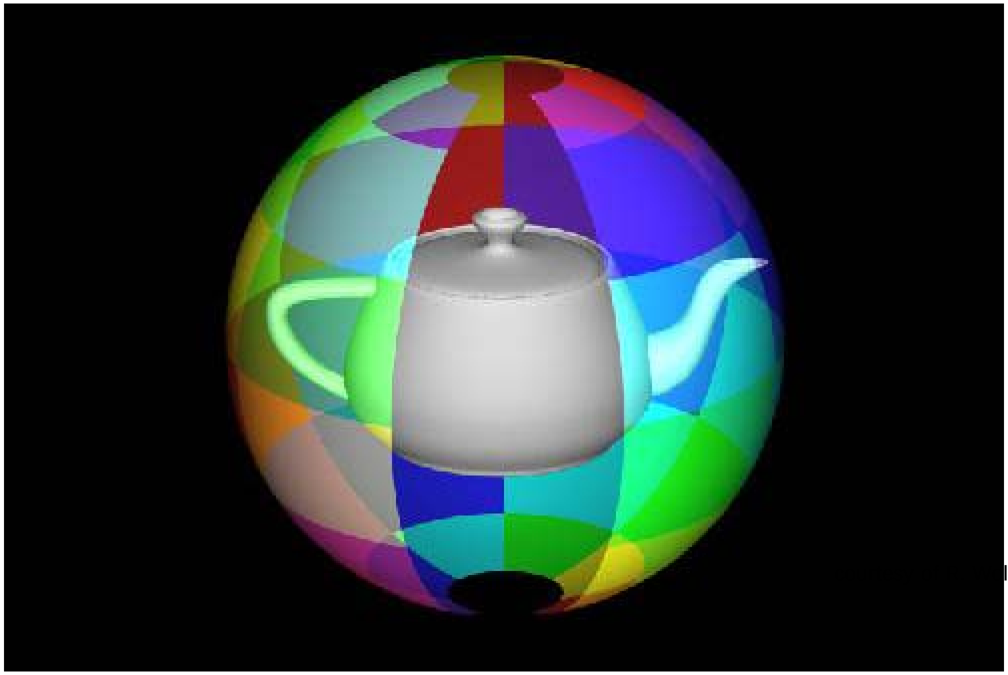
Cylindrical Projector

- Convert rectangular coordinates (x, y, z) to cylindrical (r, μ, h), use only (h, μ) to index texture image

courtesy of
R. Wolfe

Spherical Projector

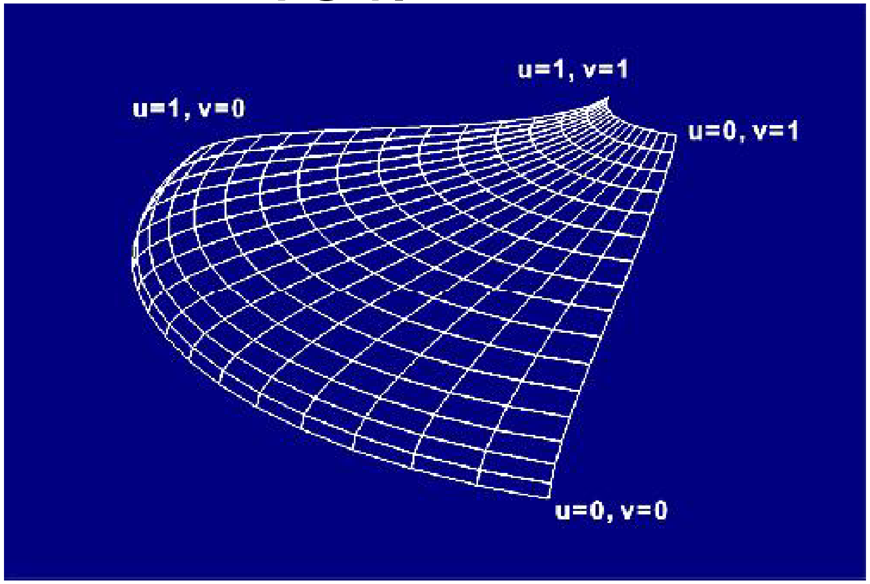
- Convert rectangular coordinates (x, y, z) to spherical (θ, ϕ)

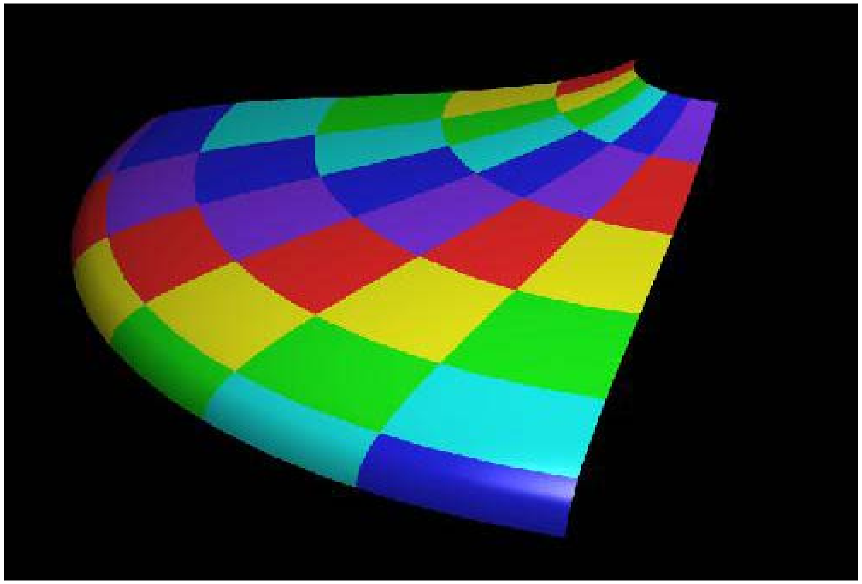


lfe

Parametric Surfaces

- A parameterized surface patch
  - $x = f(u, v)$, $y = g(u, v)$, $z = h(u, v)$
  - You will get to these kinds of surfaces in CSE 784.

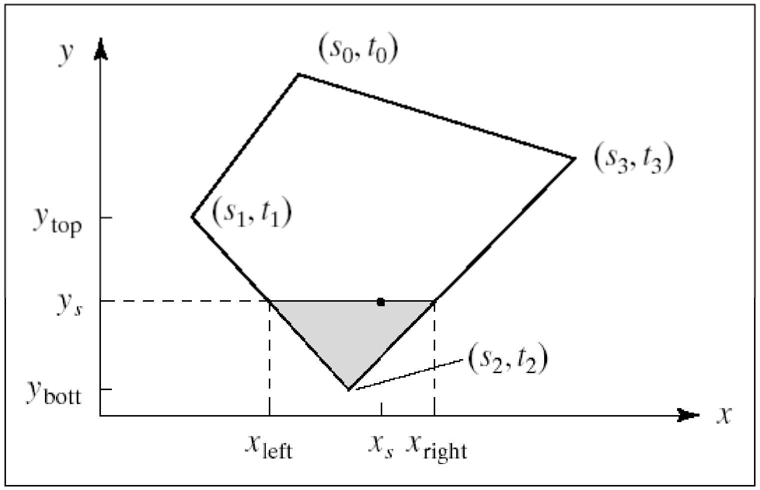




courtesy of R. Wolfe

Texture Rasterization

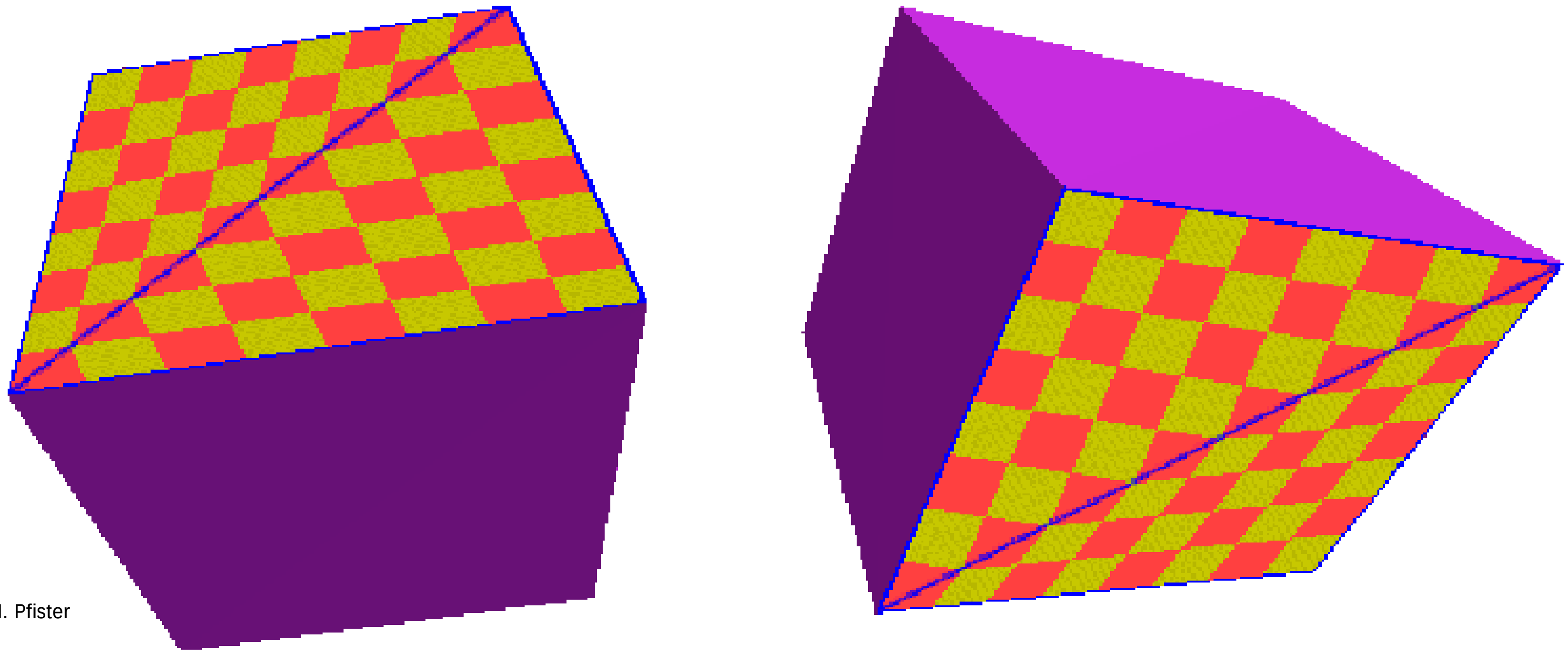
- Texture coordinates are interpolated from polygon vertices just like ... remember ...
  - Color : Gouraud shading
  - Depth: Z-buffer
  - First along polygon edges between vertices
  - Then along scanlines between left and right sides



from Hill

Linear Texture Coordinate Interpolation

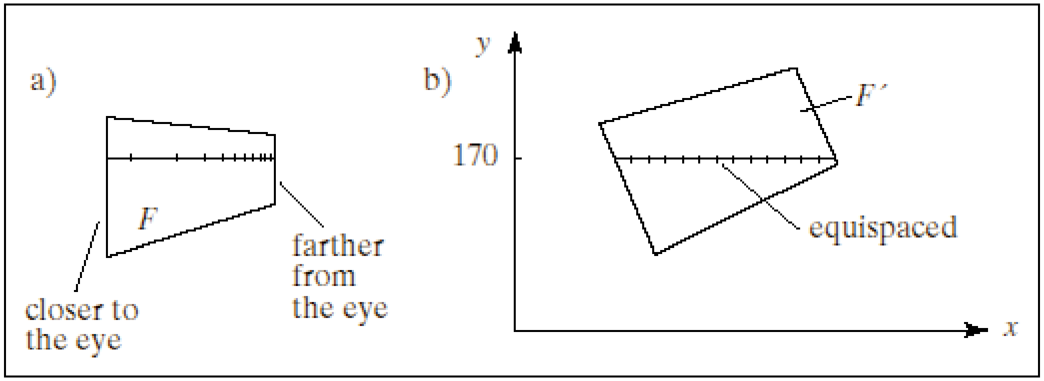
- This doesn't work in perspective projection!
- The textures look warped along the diagonal
- Noticeable during an animation



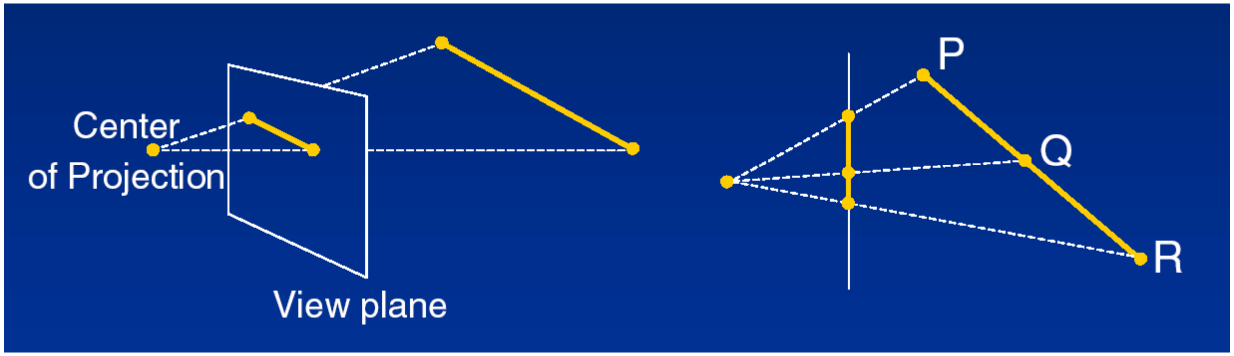
courtesy of H. Pfister

Why?

- Equal spacing in screen (pixel) space is **<u>not</u>** the same as in texture space in perspective projection
  - **Perspective foreshortening**



from Hill



courtesy of H. Pfister

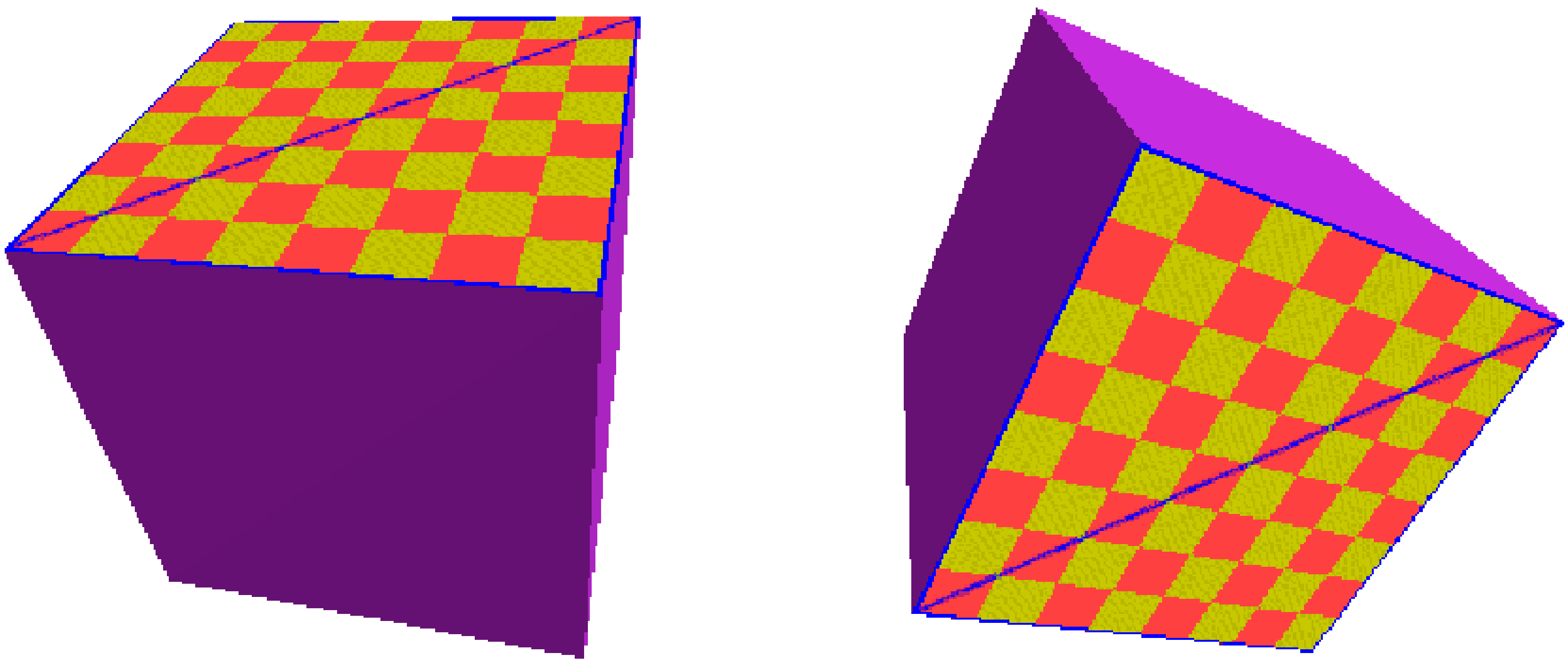
Perspective-Correct Texture Coordinate Interpolation

Interpolate (tex_coord/*w*) over the polygon, then do perspective divide after interpolation

- Compute at each vertex after perspective transformation
  - "Numerators" s/w, t/w
  - "Denominator" $1/w$
- Linearly interpolate $1/w$, s/w, and t/w across the polygon
- At each pixel
  - Perform perspective division of interpolated texture coordinates (s/w, t/w) by interpolated $1/w$ (i.e., numerator over denominator) to get (s, t)

Perspective-Correct Interpolation

- That fixed it!



Perspective Correction Hint

- Texture coordinate and color interpolation:
  - Linearly in screen space (wrong) **<u>OR</u>**
  - Persective correct interpolation (slower)

- **glHint** (GL_PERSPECTIVE_CORRECTION_HINT, **hint**), where **hint** is one of:

  - GL_NICEST: Perspective
  - GL_FASTEST: Linear
  - GL_DONT_CARE: Linear