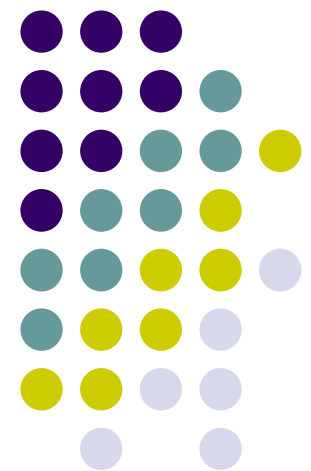


Computer Graphics (543)

Lecture 7 (Part 2): Texturing

Prof Emmanuel Agu

*Computer Science Dept.
Worcester Polytechnic Institute (WPI)*





The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second
- Many phenomena even more detailed
 - Clouds
 - Grass
 - Terrain
 - Skin
- Computationally inexpensive way to add details

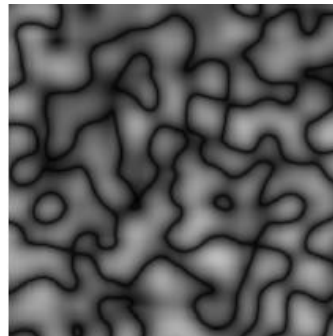
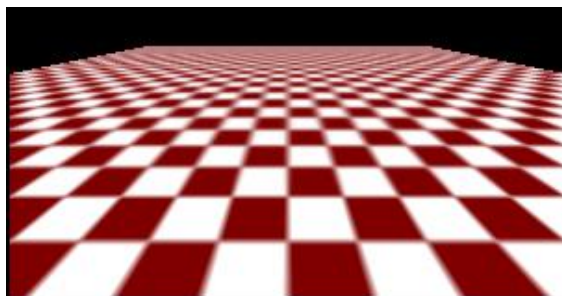


Image complexity does not affect the complexity of geometry processing (transformation, clipping...)

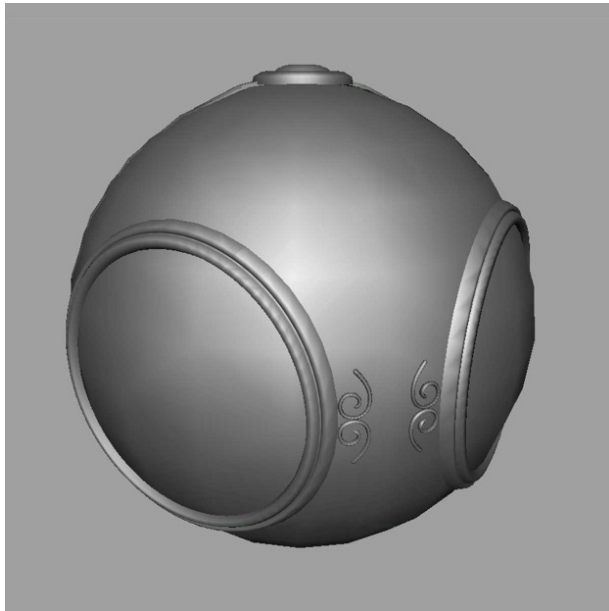
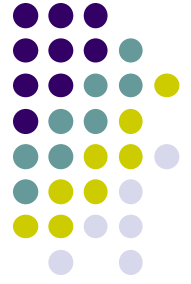


Textures in Games

- Everything is a texture except foreground characters that require interaction
- Even details on foreground texture (e.g. clothes) is texture



Types of Texturing

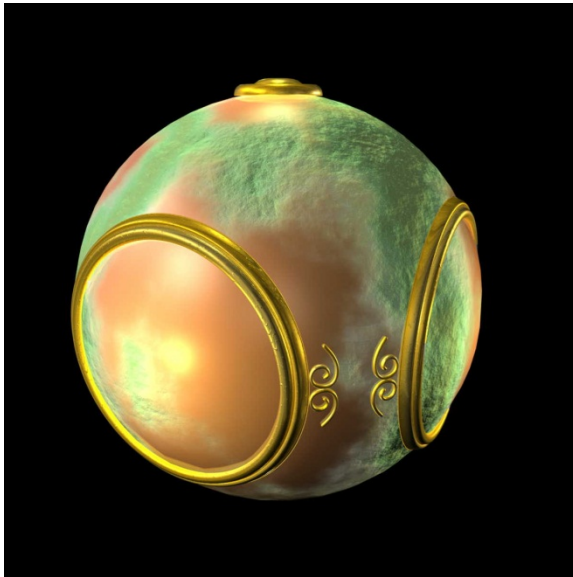


1. geometric model



2. texture mapped
Paste image (marble)
onto polygon

Types of Texturing



3. Bump mapping
Simulate surface roughness
(dimples)

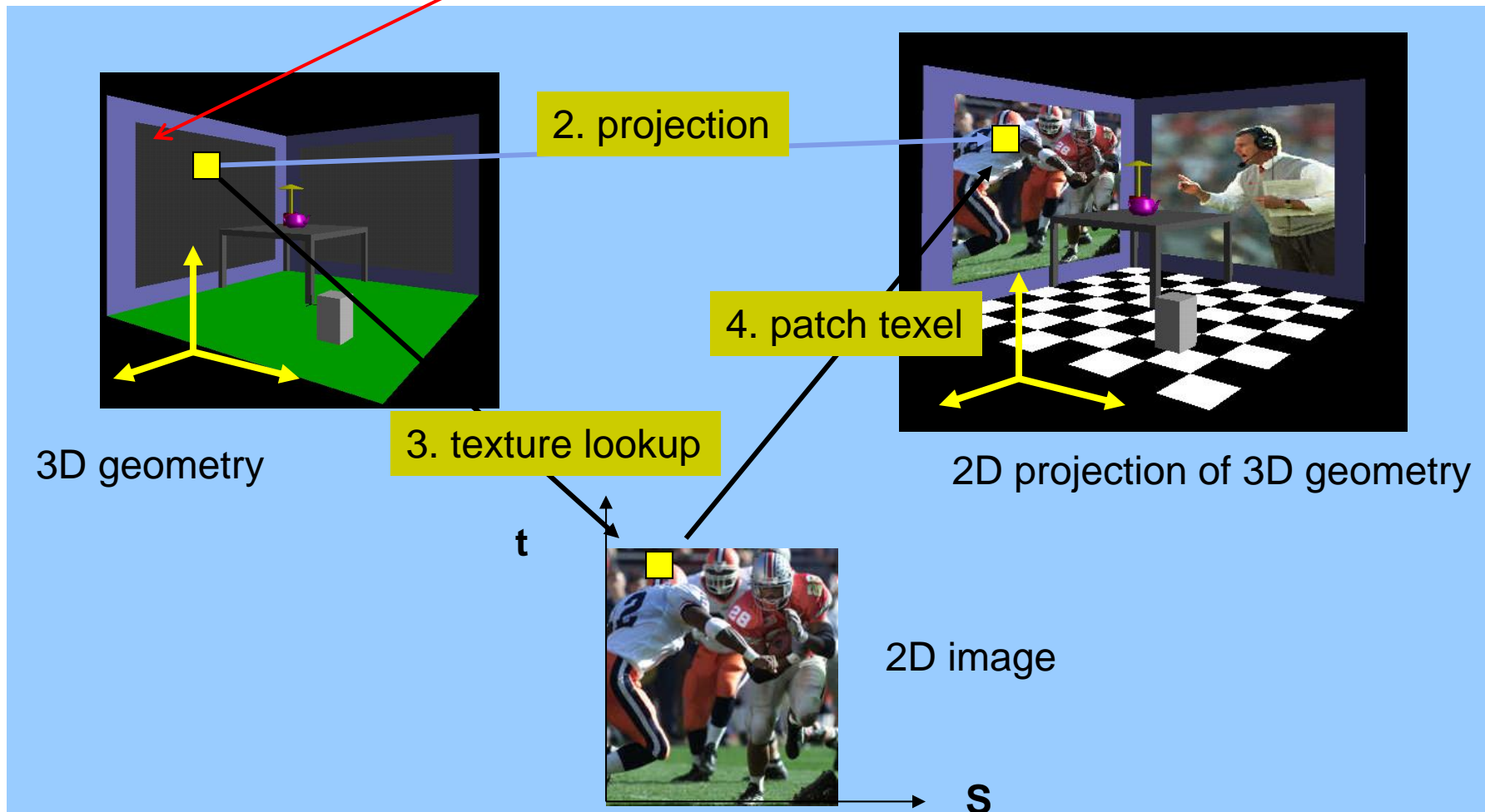


4. Environment mapping
Picture of sky/environment
over object



Texture Mapping

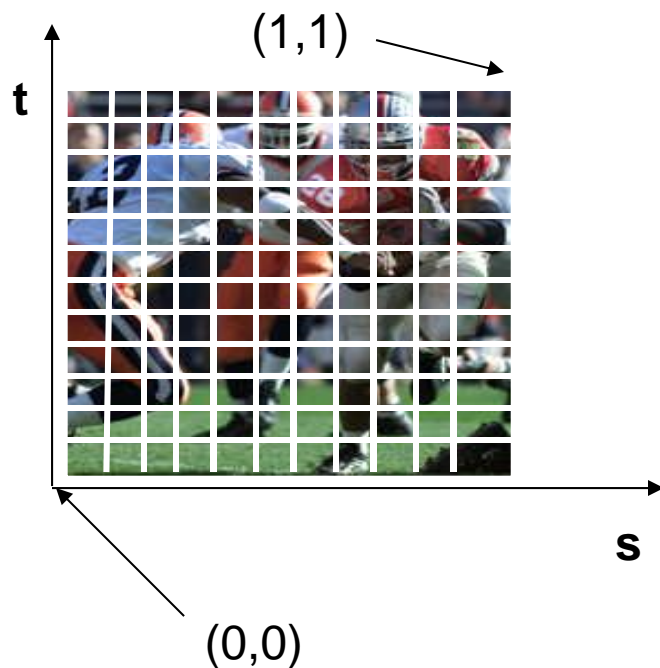
1. Define texture position on geometry





Texture Representation

- ✓ Bitmap (pixel map) textures: images (jpg, bmp, etc) loaded
- Procedural textures: E.g. fractal picture generated in .cpp file
- Textures applied in shaders



Bitmap texture:

- 2D image - 2D array **texture[height][width]**
- Each element (or **texel**) has coordinate (s, t)
- s and t normalized to [0,1] range
- Any (s,t) => [red, green, blue] color

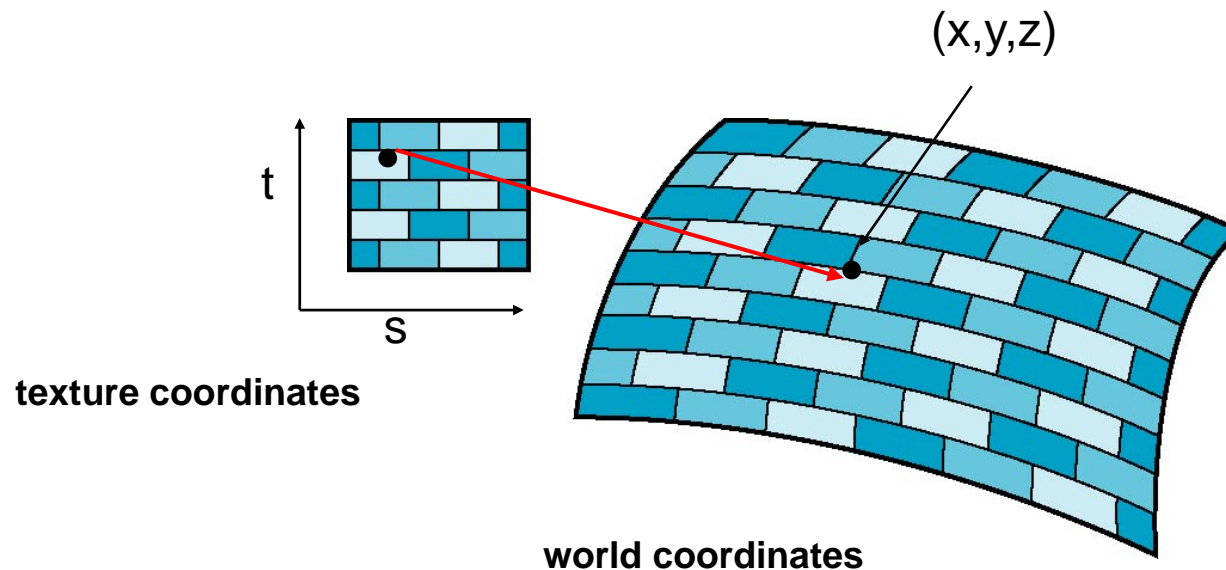


Texture Mapping

- Map? Each (x,y,z) point on object, has corresponding (s, t) point in texture

$$s = s(x,y,z)$$

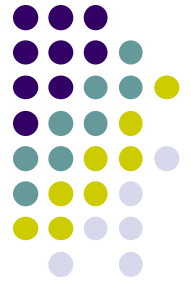
$$t = t(x,y,z)$$





6 Main Steps to Apply Texture

1. Create texture object
2. Specify the texture
 - Read or generate image
 - assign to texture (hardware) unit
 - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
 - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders



Step 1: Create Texture Object

- OpenGL has **texture objects** (multiple objects possible)
 - 1 object stores 1 texture image + texture parameters
- First set up texture object

```
GLuint mytex[1];  
glGenTextures(1, mytex); // Get texture identifier  
glBindTexture(GL_TEXTURE_2D, mytex[0]); // Form new texture object
```

- Subsequent texture functions use this object
- Another call to **glBindTexture** with new name starts new texture object



Step 2: Specifying a Texture Image

- Define input picture to paste onto geometry
- Define texture image as array of ***texels*** in CPU memory
`Glubyte my_texels[512][512][3];`
- Read in scanned images (jpeg, png, bmp, etc files)
 - If uncompressed (e.g bitmap): read into array from disk
 - If compressed (e.g. jpeg), use third party libraries (e.g. Qt, devil) to uncompress + load



← bmp, jpeg, png, etc



Step 2: Specifying a Texture Image

- Procedural texture: generate pattern in application code



- Enable texture mapping
 - `glEnable(GL_TEXTURE_2D)`
 - OpenGL supports 1-4 dimensional texture maps



Specify Image as a Texture

Tell OpenGL: this image is a texture!!

```
glTexImage2D( target, level, components,  
             w, h, border, format, type, texels );
```

target: type of texture, e.g. `GL_TEXTURE_2D`

level: used for mipmapping (0: highest resolution. More later)

components: elements per texel

w, h: width and height of `texels` in pixels

border: used for smoothing (discussed later)

format, type: describe texels

texels: pointer to texel array

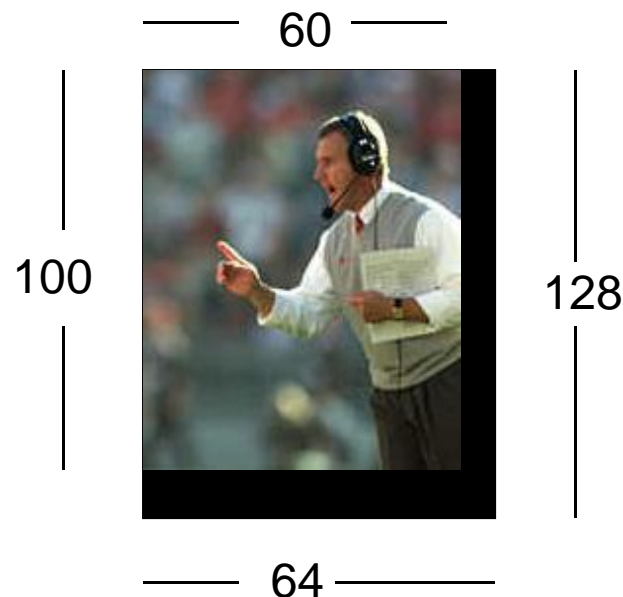
Example:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB,  
             GL_UNSIGNED_BYTE, my_texels);
```



Fix texture size

- OpenGL textures must be power of 2
- If texture dimensions not power of 2, either
 - 1) Pad zeros
 - 2) Scale the Image



Remember to adjust target polygon corners
– don't want black texels in your final picture



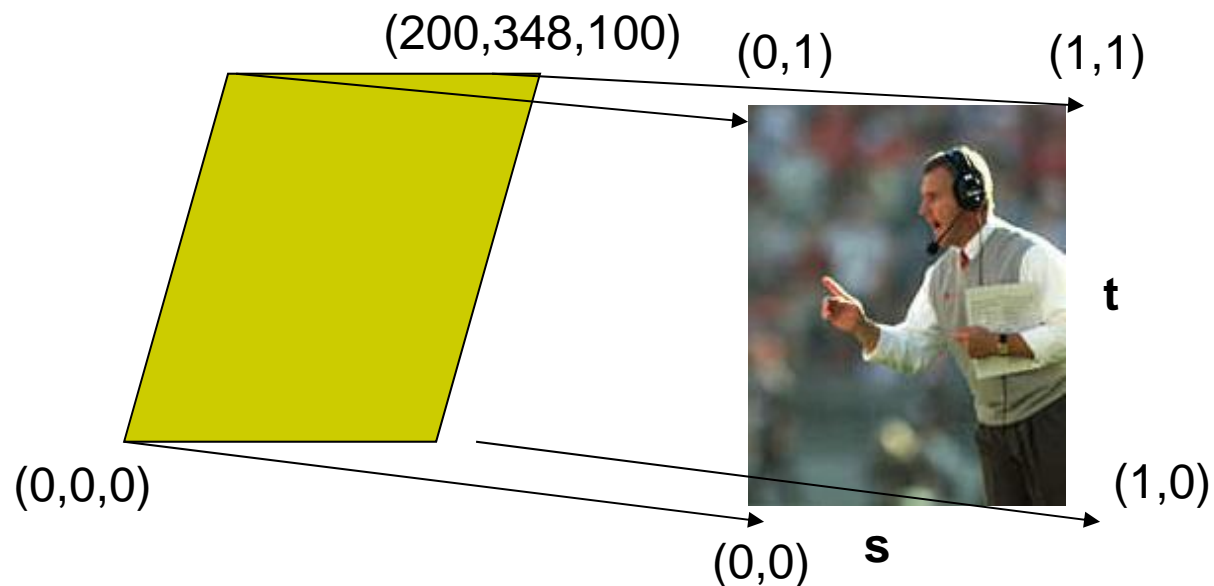
6 Main Steps. Where are we?

1. Create texture object
2. Specify the texture
 - Read or generate image
 - assign to texture (hardware) unit
 - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
 - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders



Step 3: Assign Object Corners to Texture Corners

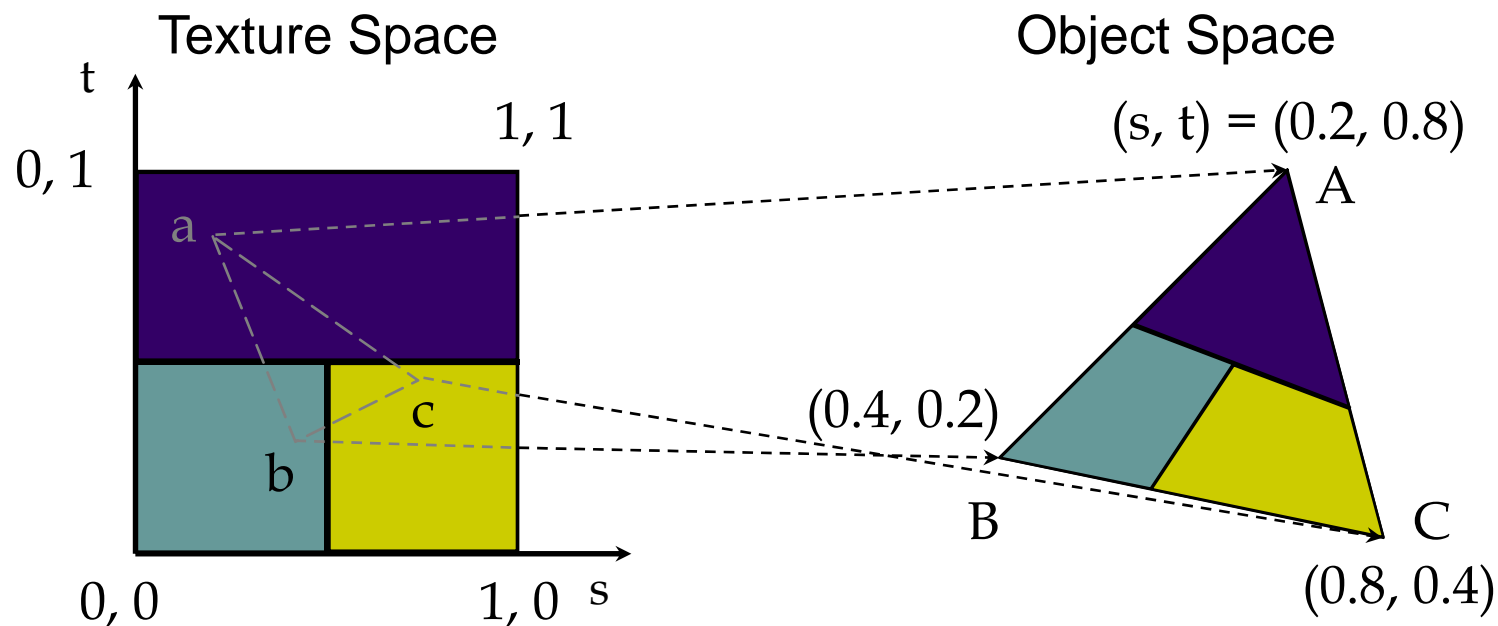
- Each object corner $(x,y,z) \Rightarrow$ image corner (s, t)
 - E.g. object $(200,348,100) \Rightarrow (1,1)$ in image
- Programmer establishes this mapping
- Target polygon can be any size/shape



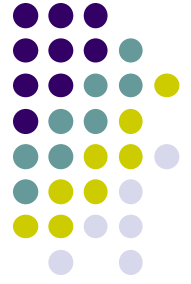


Step 3: Assigning Texture Coordinates

- After specifying corners, interior (s,t) ranges also mapped
- Example? Corners mapped below, abc subrange also mapped

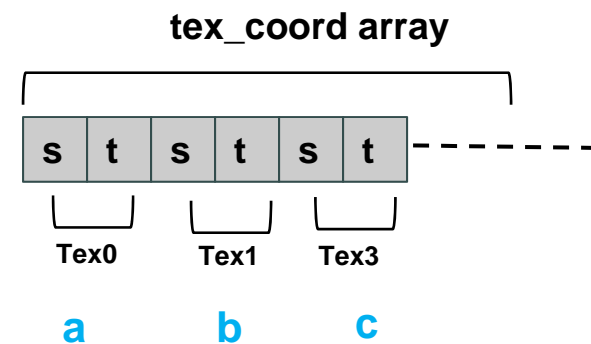
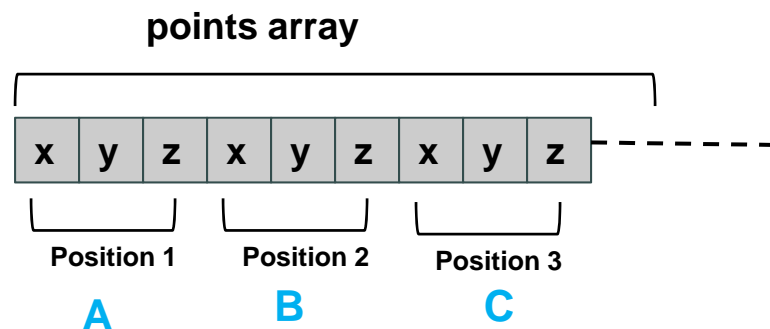


Step 3: Code for Assigning Texture Coordinates



- **Example:** Trying to map a picture to a quad
- For each quad corner (vertex), specify
 - Specify vertex (x,y,z),
 - Specify corresponding corner of texture (s, t)
- May generate array of vertices + array of texture coordinates

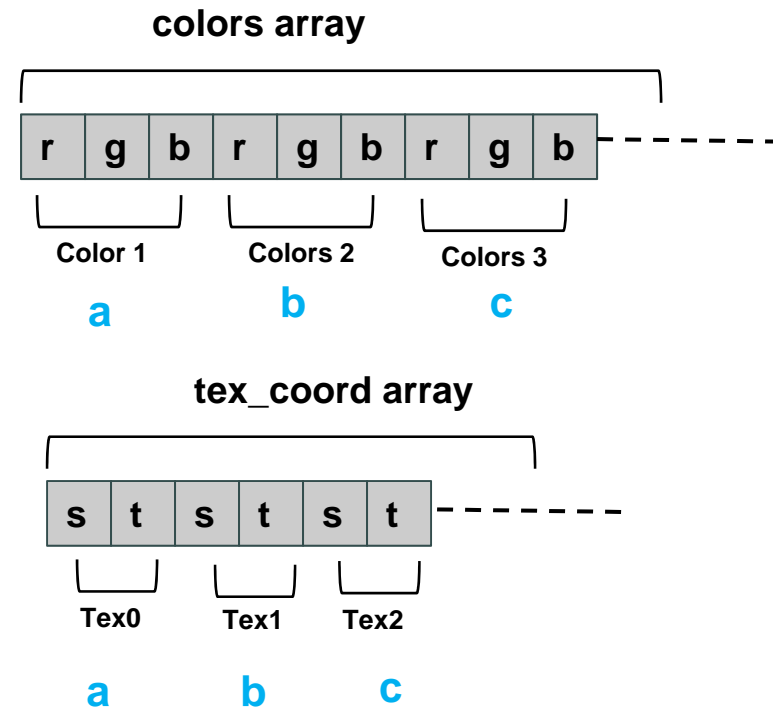
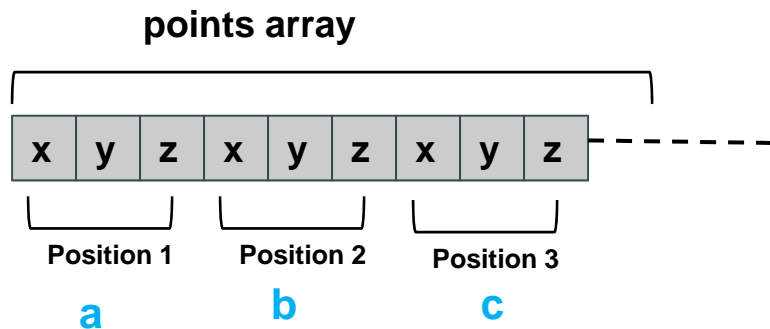
```
points[i] = point3(2,4,6);  
tex_coord[i] = point2(0.0, 1.0);
```



Step 3: Code for Assigning Texture Coordinates



```
void quad( int a, int b, int c, int d )  
{  
    quad_colors[Index] = colors[a]; // specify vertex color  
    points[Index] = vertices[a]; // specify vertex position  
    tex_coords[Index] = vec2( 0.0, 0.0 ); //specify corresponding texture corner  
    index++;  
    quad_colors[Index] = colors[b];  
    points[Index] = vertices[b];  
    tex_coords[Index] = vec2( 0.0, 1.0 );  
    Index++;  
  
    // other vertices  
}
```





Step 5: Passing Texture to Shader

- Pass vertex, texture coordinate data as vertex array
- Set texture unit

```
offset = 0;
GLuint vPosition = glGetAttribLocation( program, "vPosition" );
glEnableVertexAttribArray( vPosition );
glVertexAttribPointer( vPosition, 4, GL_FLOAT, GL_FALSE,
    0, BUFFER_OFFSET(offset) );

offset += sizeof(points);
GLuint vTexCoord = glGetAttribLocation( program, "vTexCoord" );
glEnableVertexAttribArray( vTexCoord );
glVertexAttribPointer( vTexCoord, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(offset) );

// Set the value of the fragment shader texture sampler variable
// ("texture") to the the appropriate texture unit.

glUniform1i( glGetUniformLocation(program, "texture"), 0 );
```

Variable names
in shader



Step 6: Apply Texture in Shader (Vertex Shader)



- Vertex shader receives data, output texture coordinates to fragment shader

```
in vec4 vPosition; //vertex position in object coordinates
in vec4 vColor; //vertex color from application
in vec2 vTexCoord; //texture coordinate from application
```

```
out vec4 color; //output color to be interpolated
out vec2 texCoord; //output tex coordinate to be interpolated
```

```
texCoord = vTexCoord
color = vColor
gl_Position = modelview * projection * vPosition
```

Step 6: Apply Texture in Shader (Fragment Shader)



- Textures applied in fragment shader
- Samplers return a texture color from a texture object

```
in vec4 color; //color from rasterizer
in vec2 texCoord; //texture coordinate from rasterizer
uniform sampler2D texture; //texture object from application
```

```
void main() {
    gl_FragColor = color * texture2D( texture, texCoord );
}
```

Output color
Of fragment

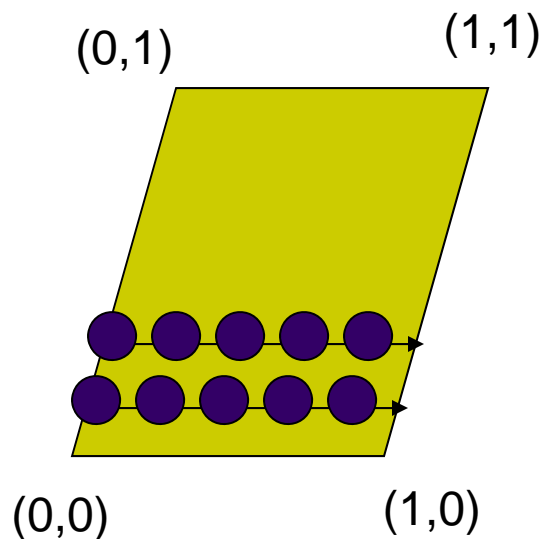
Original color
of object

Lookup color of
texCoord (s,t) in texture

Map textures to surfaces

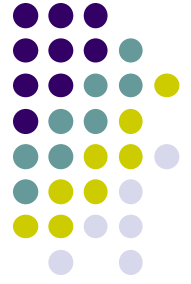


- Texture mapping is performed in rasterization

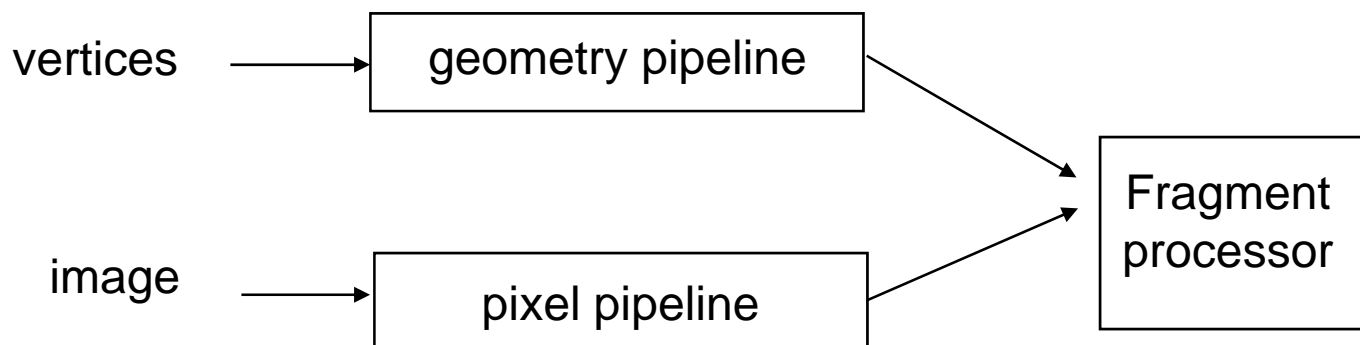


- ☐ For each pixel, its texture coordinates (s, t) interpolated based on corners' texture coordinates (why not just interpolate the color?)
- ☐ The interpolated texture (s, t) coordinates are then used to perform texture lookup

Texture Mapping and the OpenGL Pipeline



- Images and geometry flow through separate pipelines that join during fragment processing
 - Object geometry: geometry pipeline
 - Image: pixel pipeline
 - “complex” textures do not affect geometric complexity

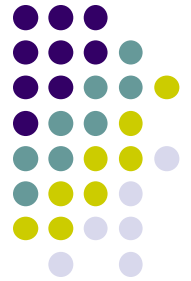




6 Main Steps to Apply Texture

1. Create texture object
2. Specify the texture
 - Read or generate image
 - assign to texture (hardware) unit
 - enable texturing (turn on)
3. Assign texture (corners) to Object corners
4. Specify texture parameters
 - wrapping, filtering
5. Pass textures to shaders
6. Apply textures in shaders

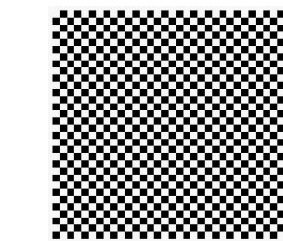
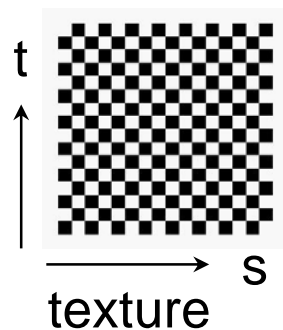
still haven't talked
about setting texture
parameters



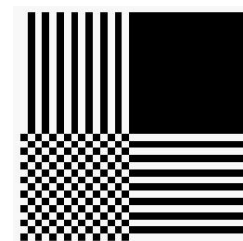
Step 4: Specify Texture Parameters

- Texture parameters control how texture is applied
 - **Wrapping parameters** used if s, t outside $(0,1)$ range
 - Clamping:** if $s, t > 1$ use 1, if $s, t < 0$ use 0
 - Wrapping:** use s, t modulo 1

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP )  
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT )
```



GL_REPEAT



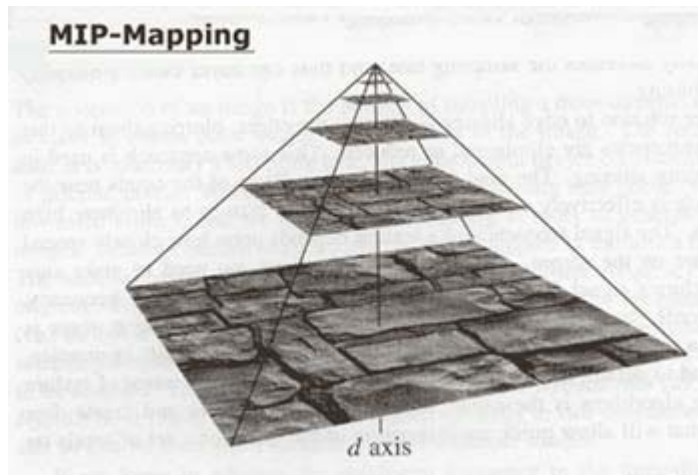
GL_CLAMP

Step 4: Specify Texture Parameters

Mipmapped Textures



- **Mipmapping** pre-generates prefiltered (averaged) texture maps of decreasing resolutions
- Declare mipmap level during texture definition
`glTexImage2D(GL_TEXTURE_2D, level, ...)`



Step 4: Specify Texture Parameters

Environment Parameters



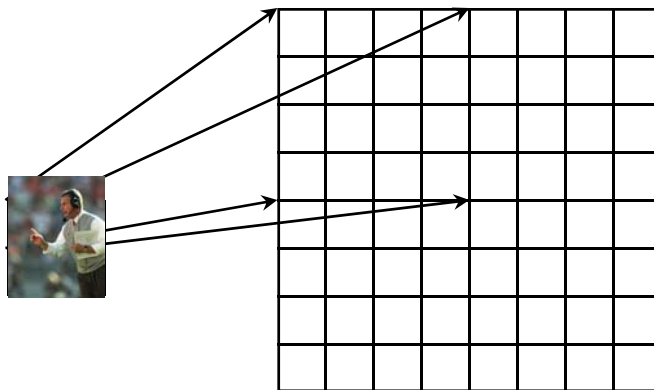
- **Environment** parameters determine how texture combines with original surface
 - `glTexEnv{fi}[v](GL_TEXTURE_ENV, prop, param)`
- `GL_TEXTURE_ENV_MODE` modes
 - `GL_MODULATE`: multiply texture and object color
 - `GL_BLEND`: linear combination of texture and object color
 - `GL_REPLACE`: use only texture color
 - `GL(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);`
- E.g: `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`



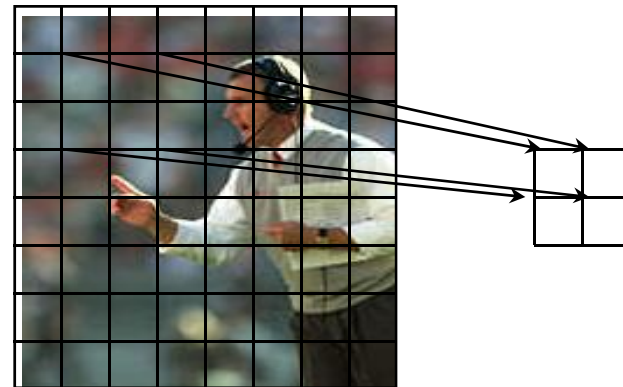
Magnification and Minification

Magnification: Stretch small texture to fill many pixels

Minification: Shrink large texture to fit few pixels



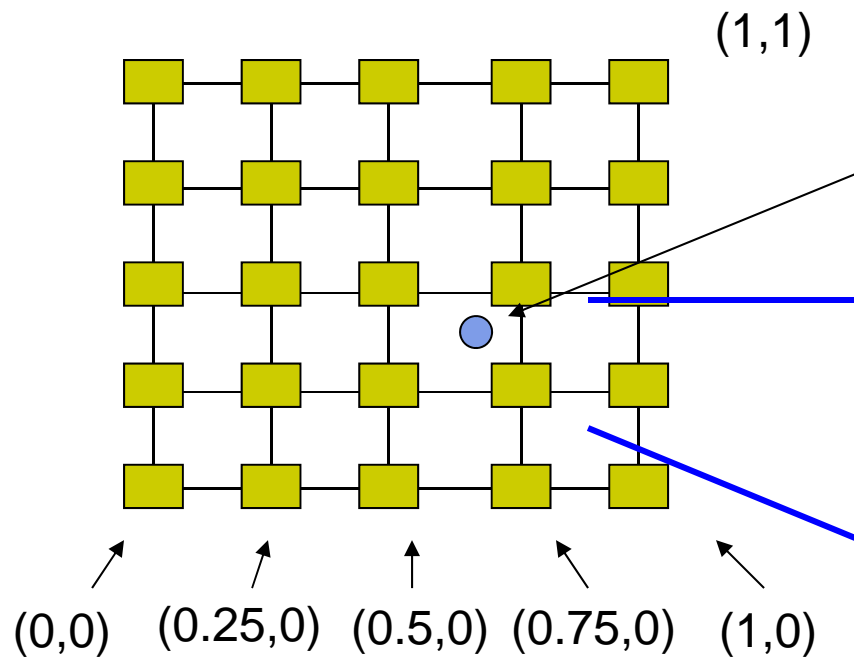
Texture Polygon
Magnification



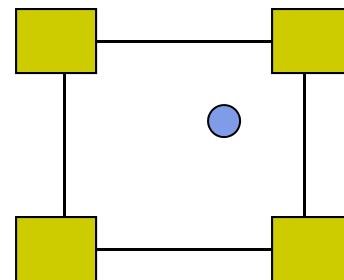
Texture Polygon
Minification

Step 4: Specify Texture Parameters

Texture Value Lookup



How about coordinates that are not exactly at the intersection (pixel) positions?



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



Example: Texture Magnification

- 48 x 48 image projected (stretched) onto 320 x 320 pixels

Nearest neighbor filter



Bilinear filter
(avg 4 nearest texels)



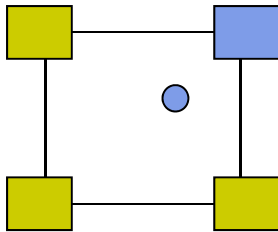
Cubic filter
(weighted avg. 5 nearest texels)





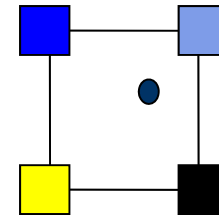
Texture mapping parameters

- 1) Nearest Neighbor (lower image quality)



```
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

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



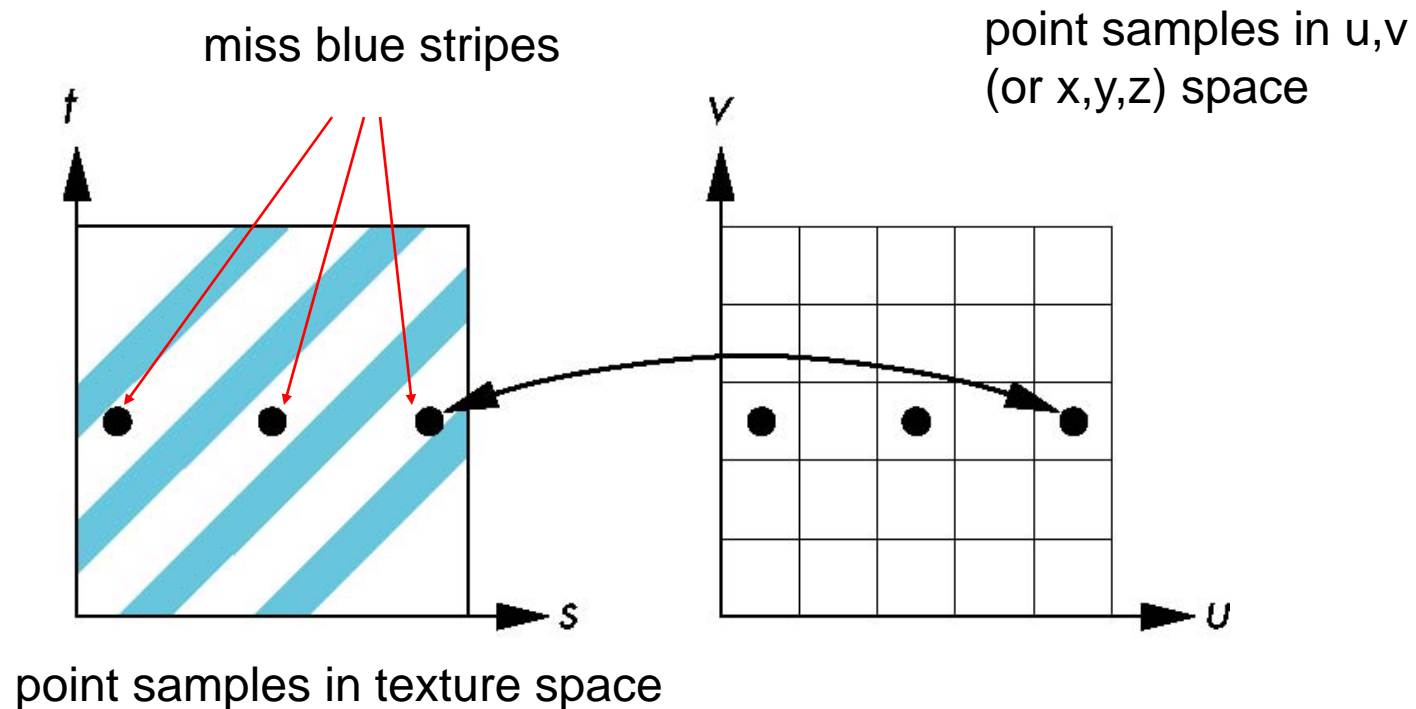
```
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_MIN_FILTER,  
GL_LINEAR)
```

Or GL_TEXTURE_MAX_FILTER



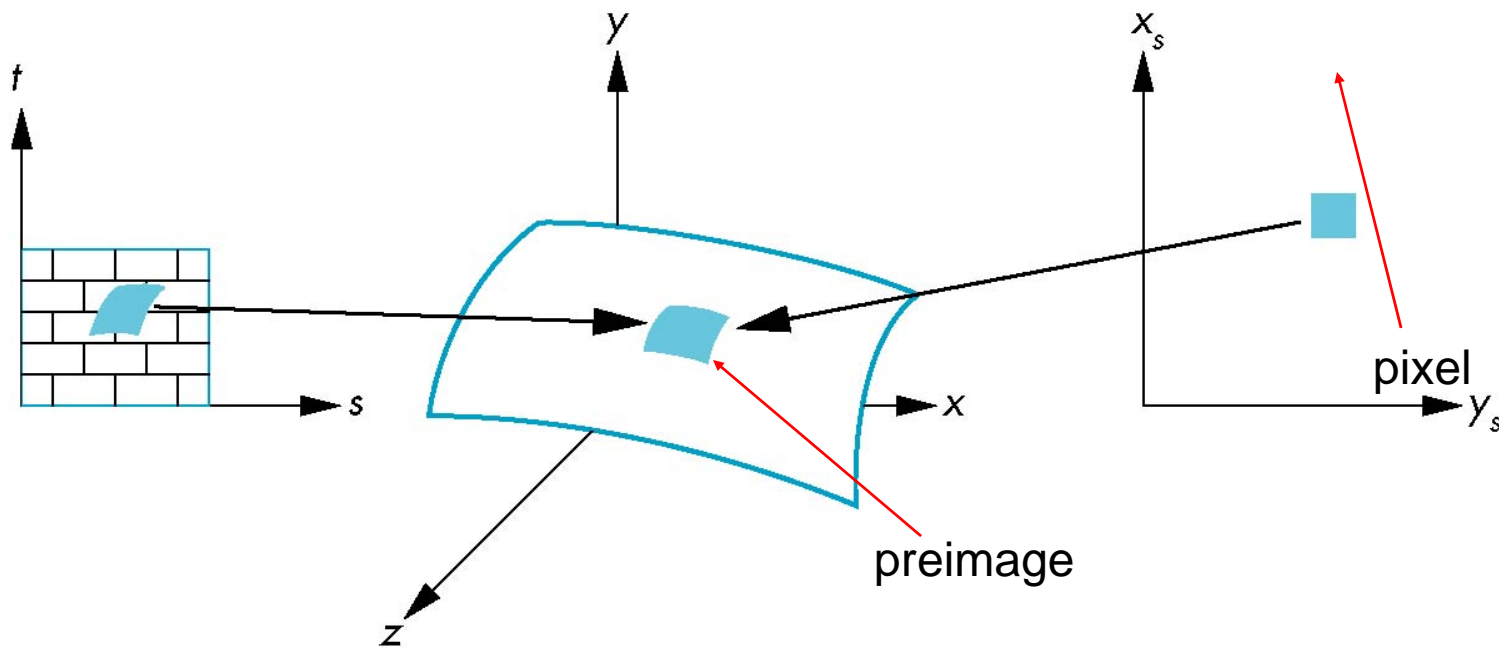
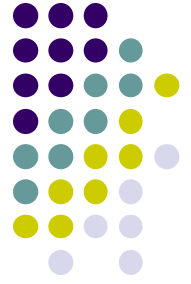
Dealing with Aliasing

- Point sampling of texture can lead to aliasing errors



Area Averaging

Better but slower option is *area averaging*





Other Stuff

- Wrapping texture onto curved surfaces. E.g. cylinder, can, etc

$$s = \frac{\theta - \theta_a}{\theta_b - \theta_a}$$

$$t = \frac{z - z_a}{z_b - z_a}$$

- Wrapping texture onto sphere

$$s = \frac{\theta - \theta_a}{\theta_b - \theta_a}$$

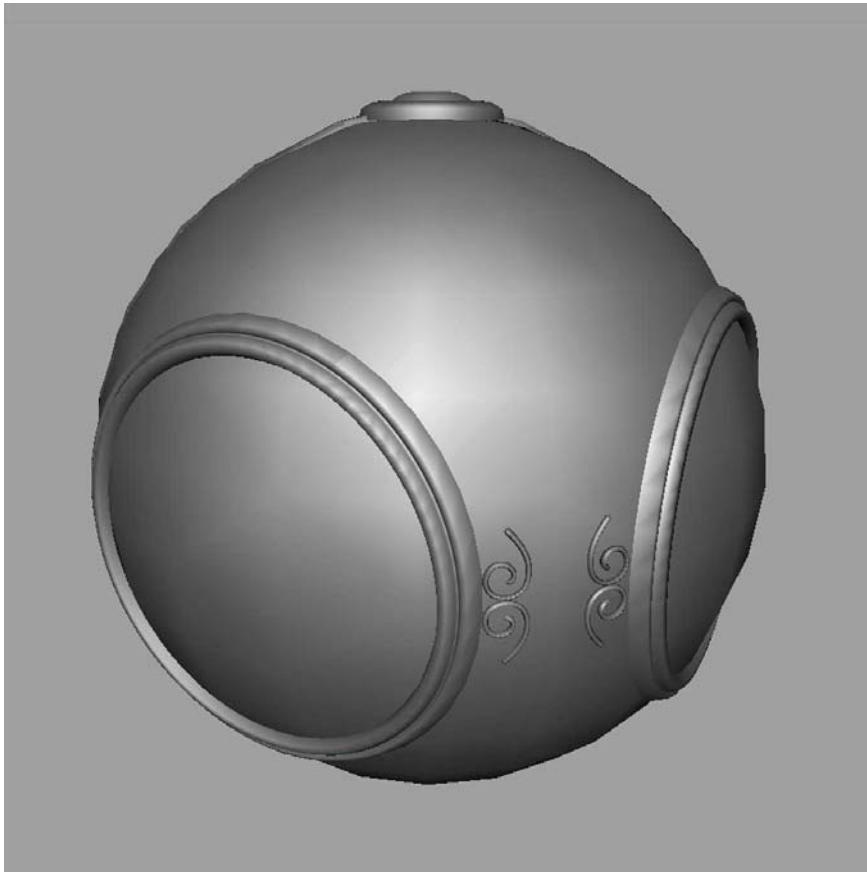
$$t = \frac{\phi - \phi_a}{\phi_b - \phi_a}$$

- Bump mapping: perturb surface normal by a quantity proportional to texture

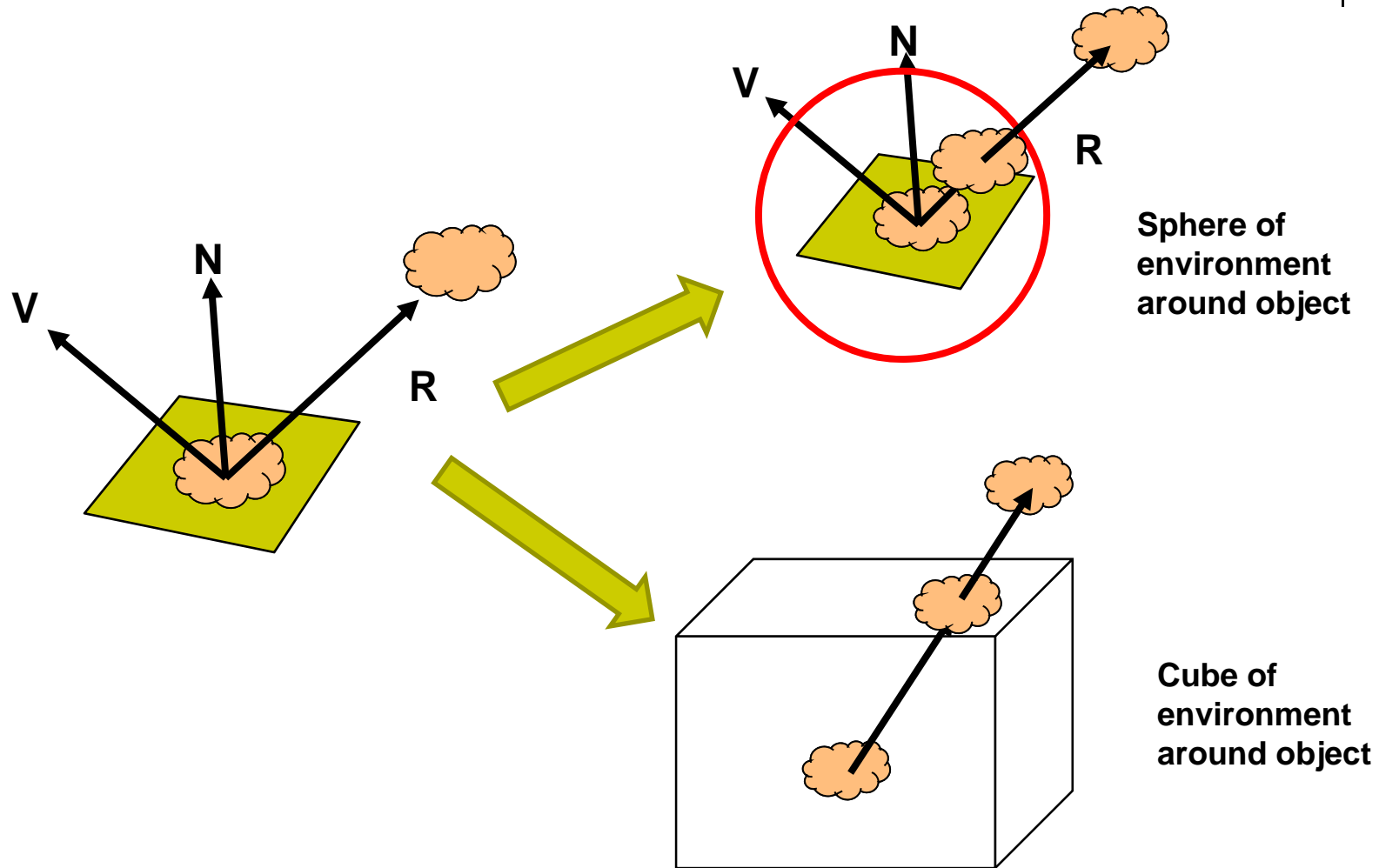


Environment Mapping

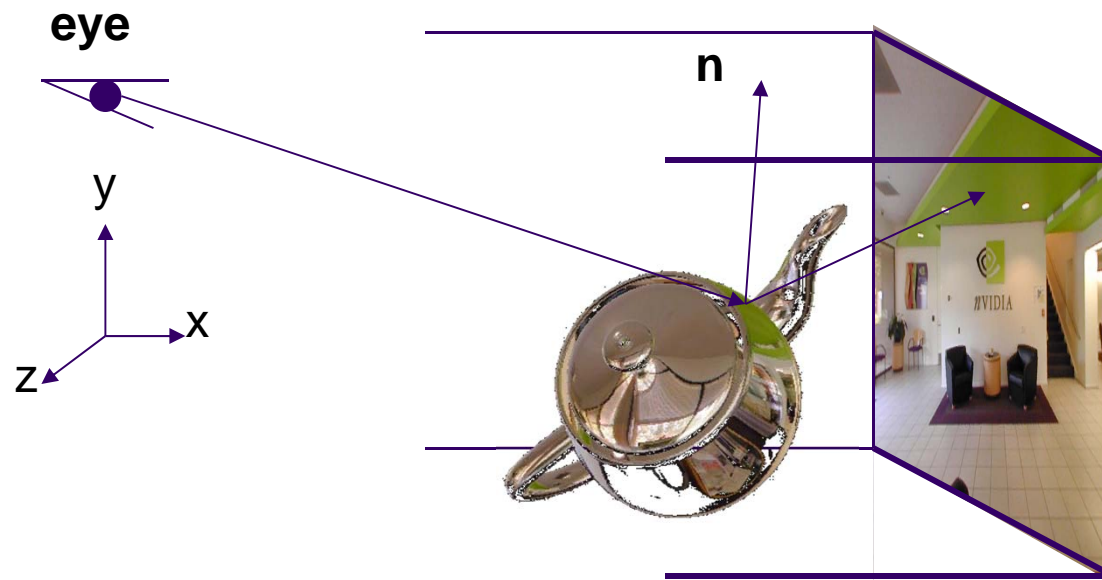
- Environmental mapping is way to create the appearance of highly reflective surfaces



Reflecting the Environment

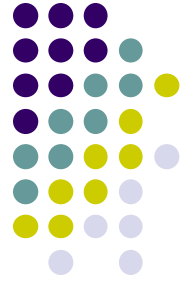


Cube mapping

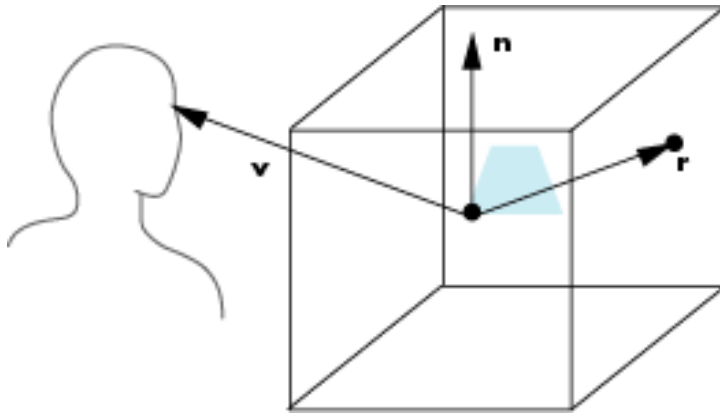


- Need to compute reflection vector, \mathbf{r}
- Use \mathbf{r} by for lookup
- OpenGL: hardware support for cube maps

Environment Map



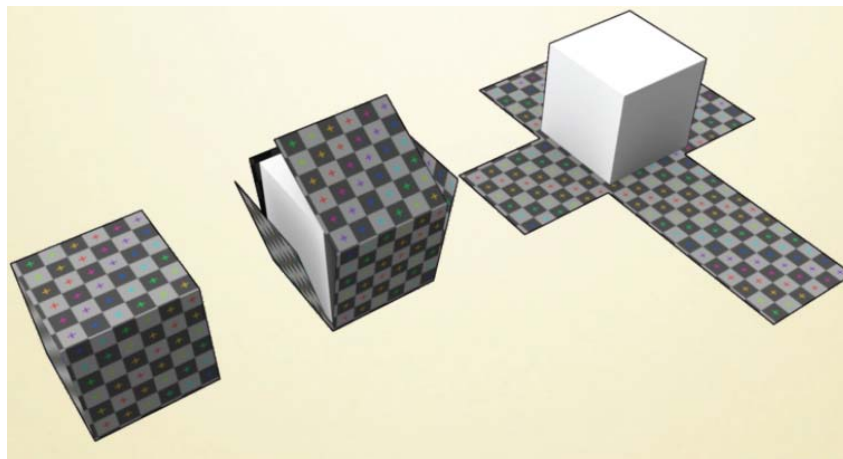
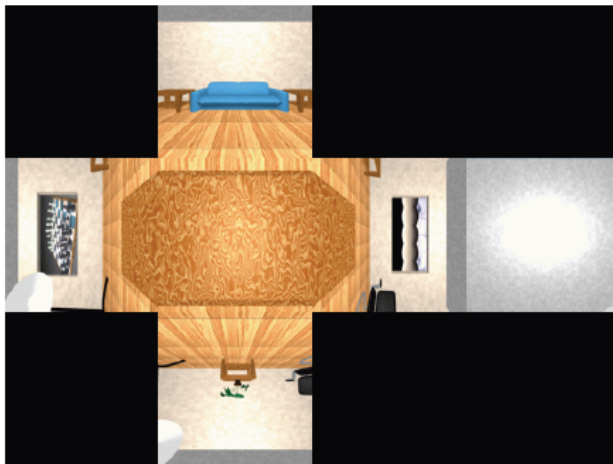
Use reflection vector to locate texture in cube map





Cube Environment Map Example

- Six textures: one for each face cube surrounding object
- Load 6 textures separately into 1 OpenGL cubemap



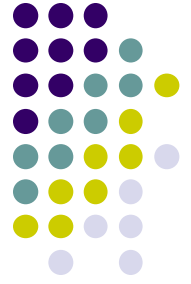
Cube Maps



- Loaded cube map texture can be accessed in GLSL through cubemap sampler

```
vec4 texColor = textureCube(mycube, texcoord);
```

- Texture coordinates must be 3D



References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley