Computer Graphics (543)
Lecture 10 (Part 2): Texturing

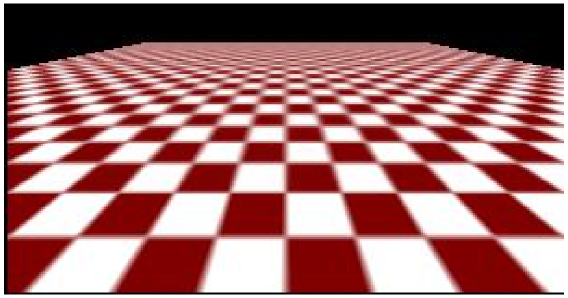
Prof Emmanuel Agu

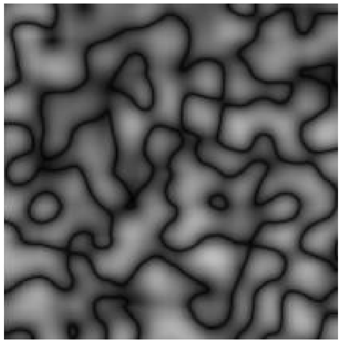
Computer Science Dept.

Worcester Polytechnic Institute (WPI)

The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin
- Computationally inexpensive way to add details





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

Modeling an Orange

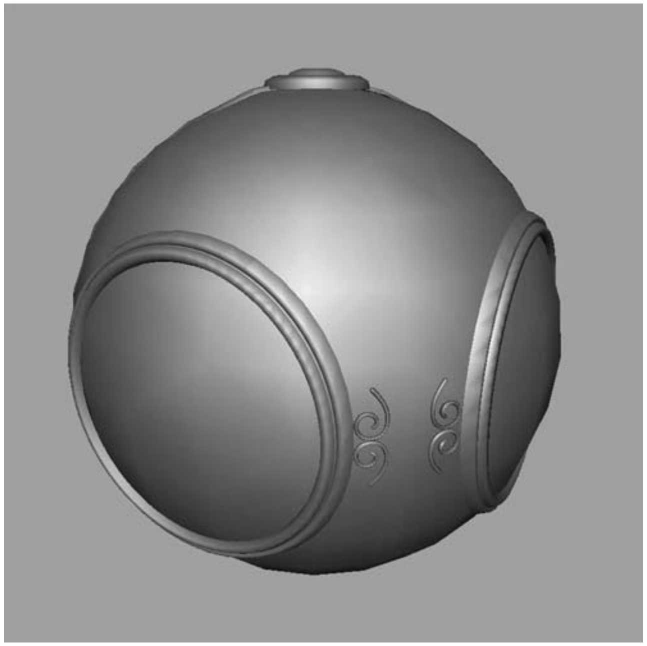
- Consider problem of modeling an orange (the fruit)
- Start with an orange-colored sphere: Too simple
- Replace sphere with a more complex shape
  - Does not capture surface characteristics (dimples)
  - Takes too many polygons to model all the dimples

Modeling an Orange (2)

- Take a picture of a real orange, scan it, and “paste” onto simple geometric model
  - Known as texture mapping
- Still might not be sufficient because resulting surface will be smooth
  - Simulate surface roughness: bump mapping

Three Types of Mapping

- Texture Mapping
  - Paste image onto polygon

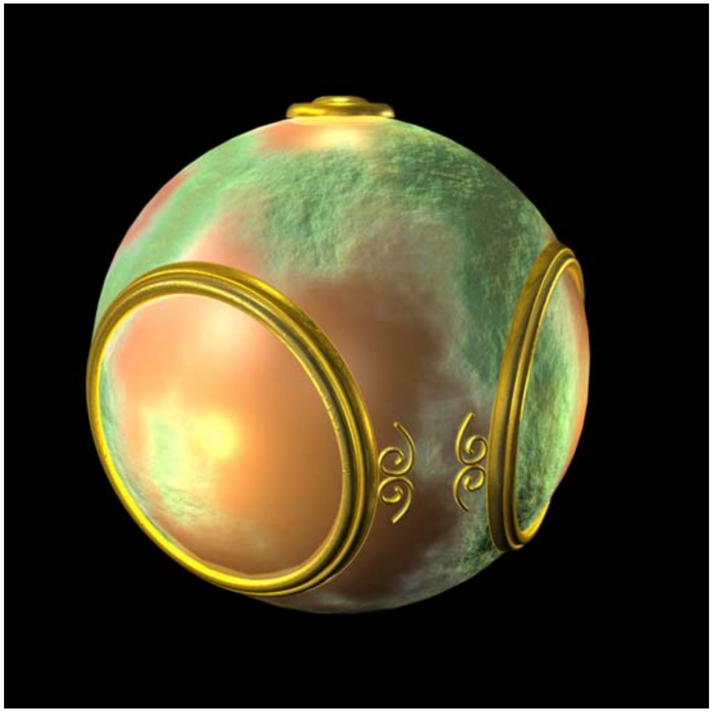


geometric model

texture mapped

Three Types of Mapping

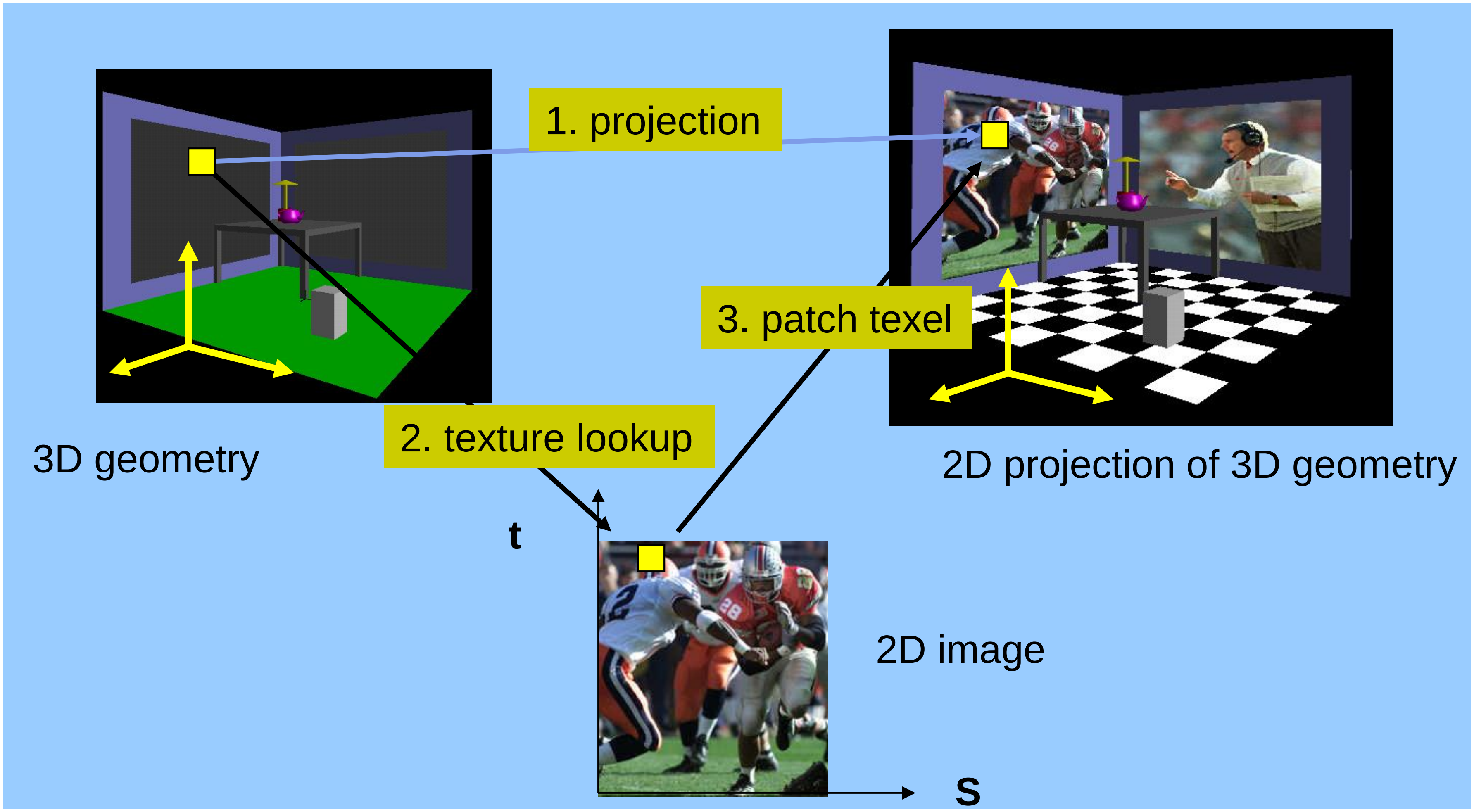
- Bump mapping
  - Alters normal vectors during rendering process to simulate surface roughness



Three Types of Mapping

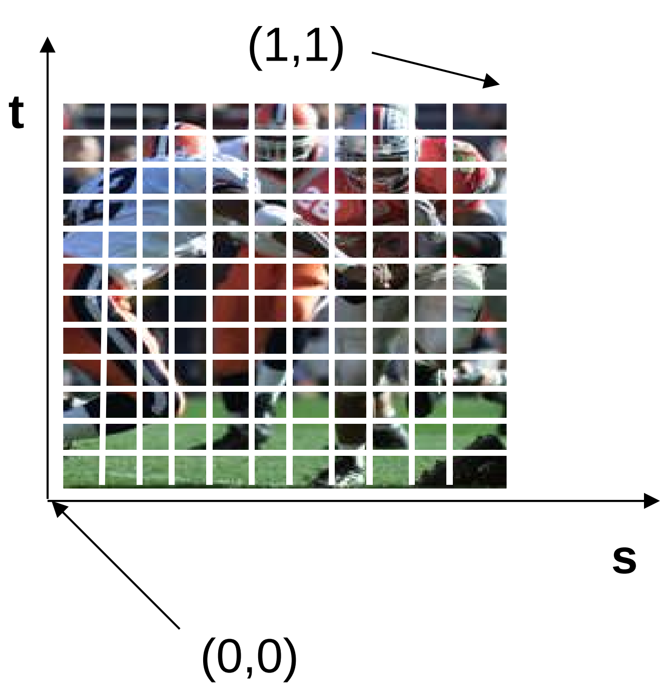
- Environment (reflection mapping)
  - Uses picture of the sky/environment for texture maps

Texture Mapping



Texture Representation

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

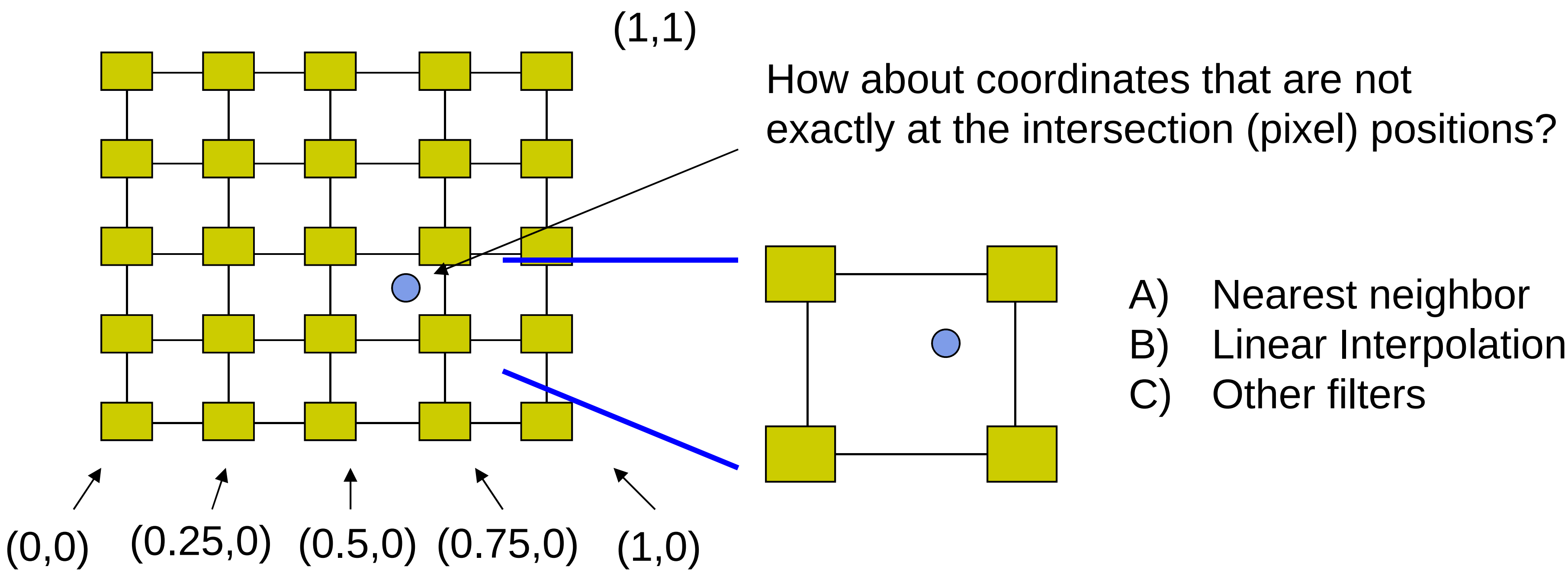


Bitmap texture:

- 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 also a unique image value (i.e., a unique [red, green, blue] set)

Texture Value Lookup

- For given texture coordinates (s,t), we can find a unique image value from the texture map



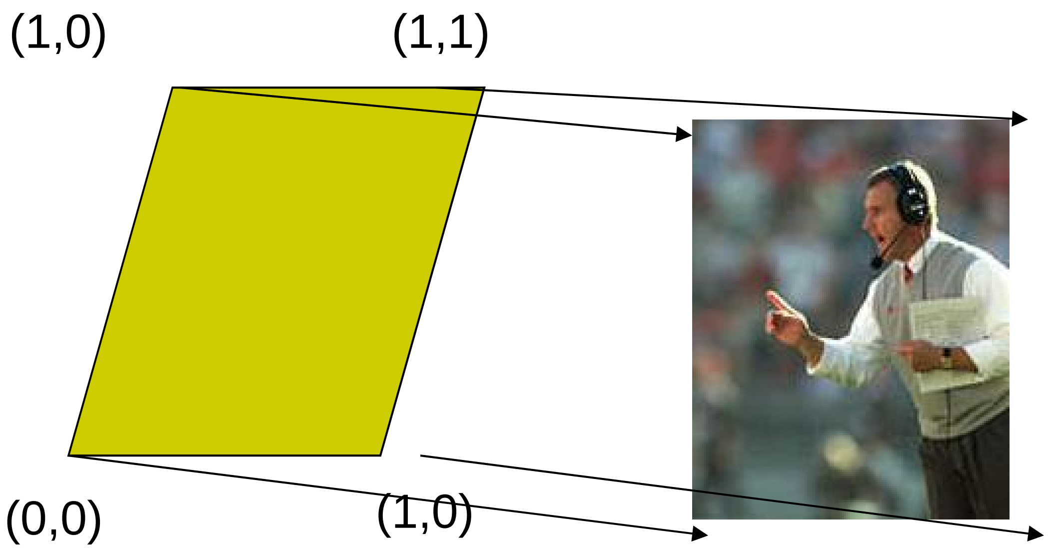
A) Nearest neighbor
B) Linear Interpolation
C) Other filters

Map textures to surfaces

- Establish mapping from texture to surfaces (polygons):

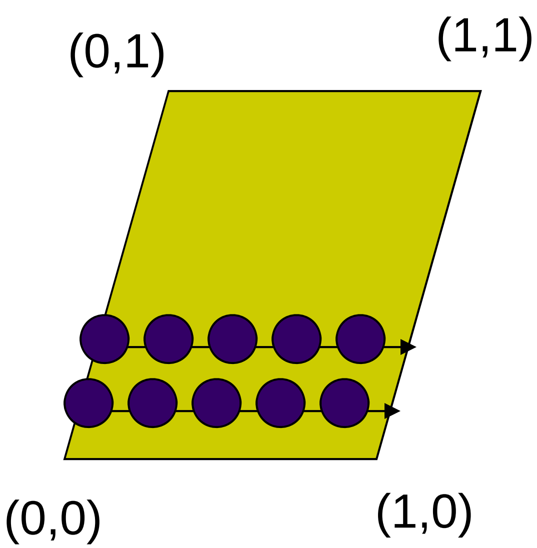
  - Application program needs to specify texture coordinates for each corner of the polygon

The polygon can be in an arbitrary size



Map textures to surfaces

- Texture mapping is performed in rasterization

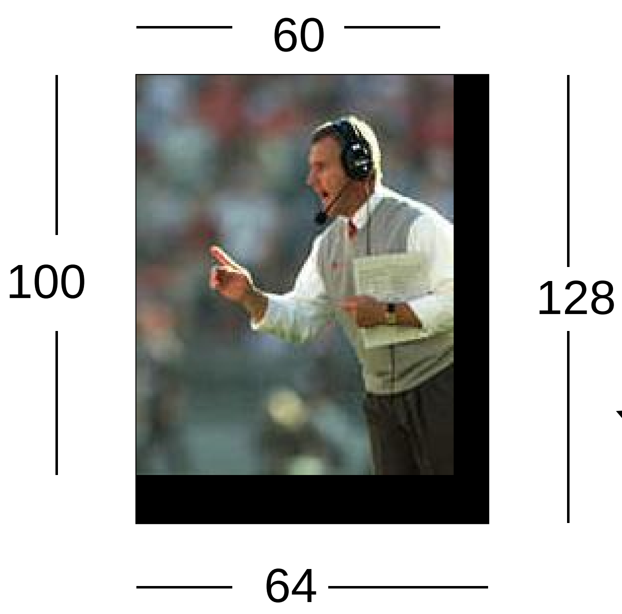


❑ For each pixel that is to be painted, its texture coordinates (s, t) are determined (interpolated) based on the corners' texture coordinates (why not just interpolate the color?)

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

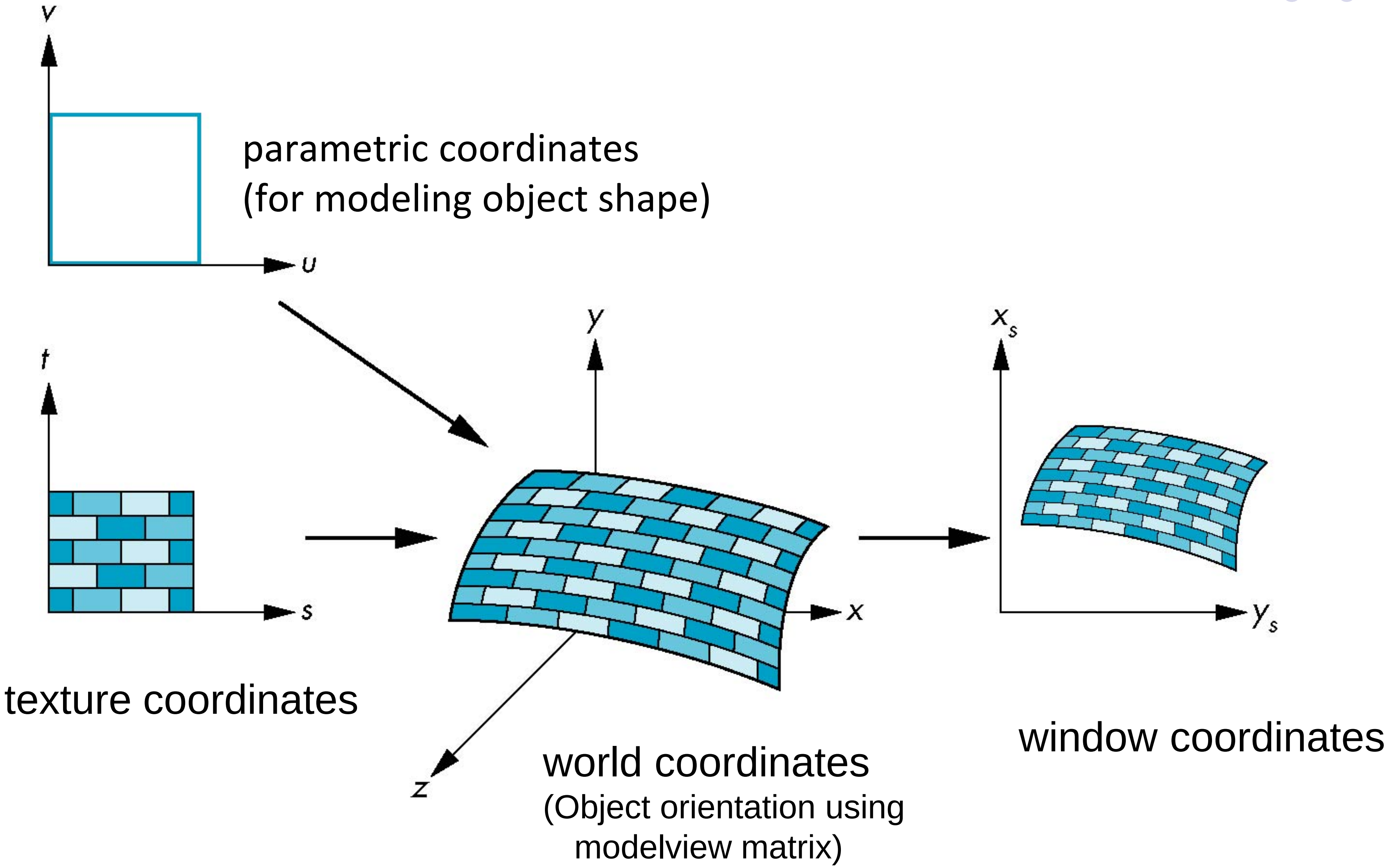
Fix texture size

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



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

Texture Mapping

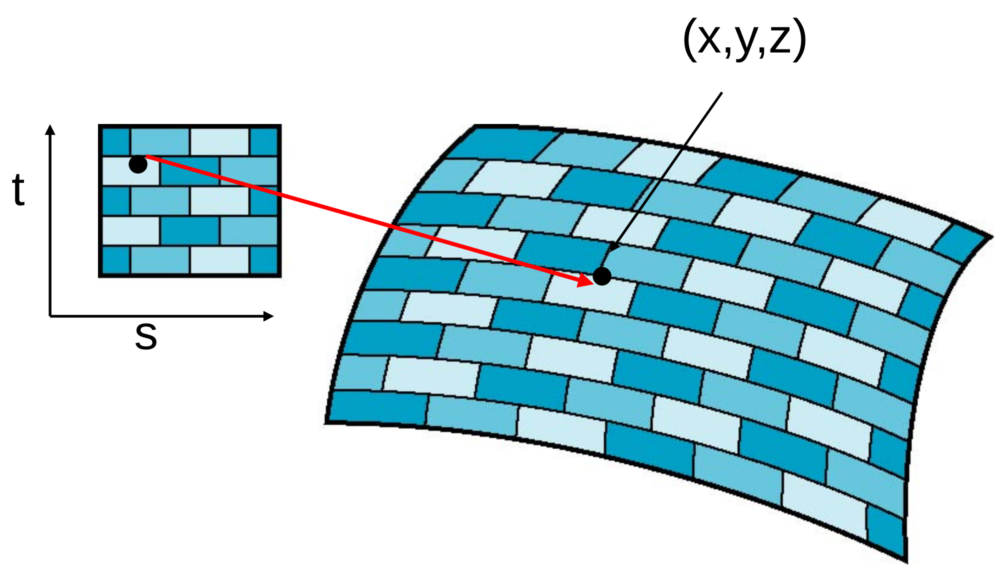


Texture Mapping

- Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form

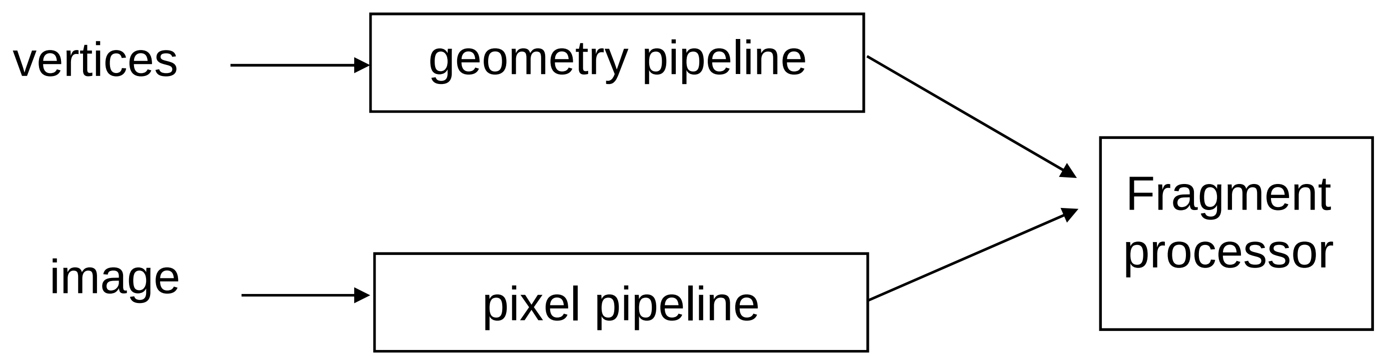
$s = s(x,y,z)$

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



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



Basic Stragegy

Three steps to applying a texture

1. specify the texture
   - Read or generate image
   - assign to texture
   - enable texturing
2. assign texture coordinates to vertices
   - Proper mapping function is left to application
3. specify texture parameters
   - wrapping, filtering

2D Texture Mapping

- OpenGL has **texture objects** for storing each texture image + texture parameters
- So, first step is to setting up texture object

```
GLuint mytex[1];
glGenTextures(1, mytex);              // Get new 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 1: Specifying a Texture Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory

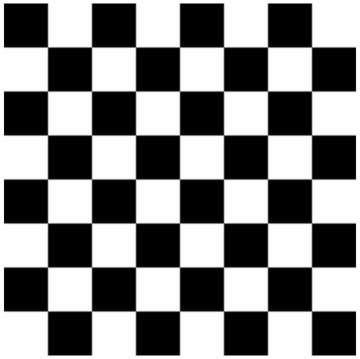
```
Glubyte my_texels[512][512][3];
```

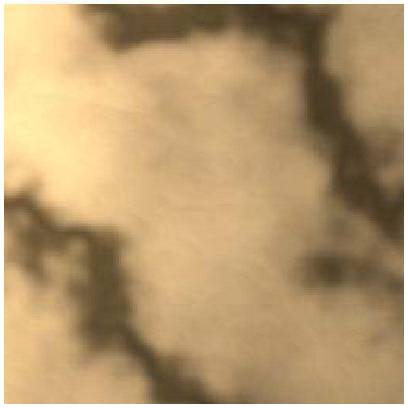
- Define as any other pixel map
  - Read in scanned images (jpeg, png, bmp, etc files)
  - If uncompressed (e.g bitmap), read in directly from disk into array
  - If compressed (e.g. jpeg), you may use third party tools like Qt or devil

bmp, jpeg, png, etc

Step 1: Specifying a Texture Image

- Define as any other pixel map
  - Alternatively generate by application code. E.g. procedural texture





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

Define Image as 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 (discussed later)

`components:` elements per texel

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

`border:` used for smoothing (discussed later)

`format and type:` describe texels

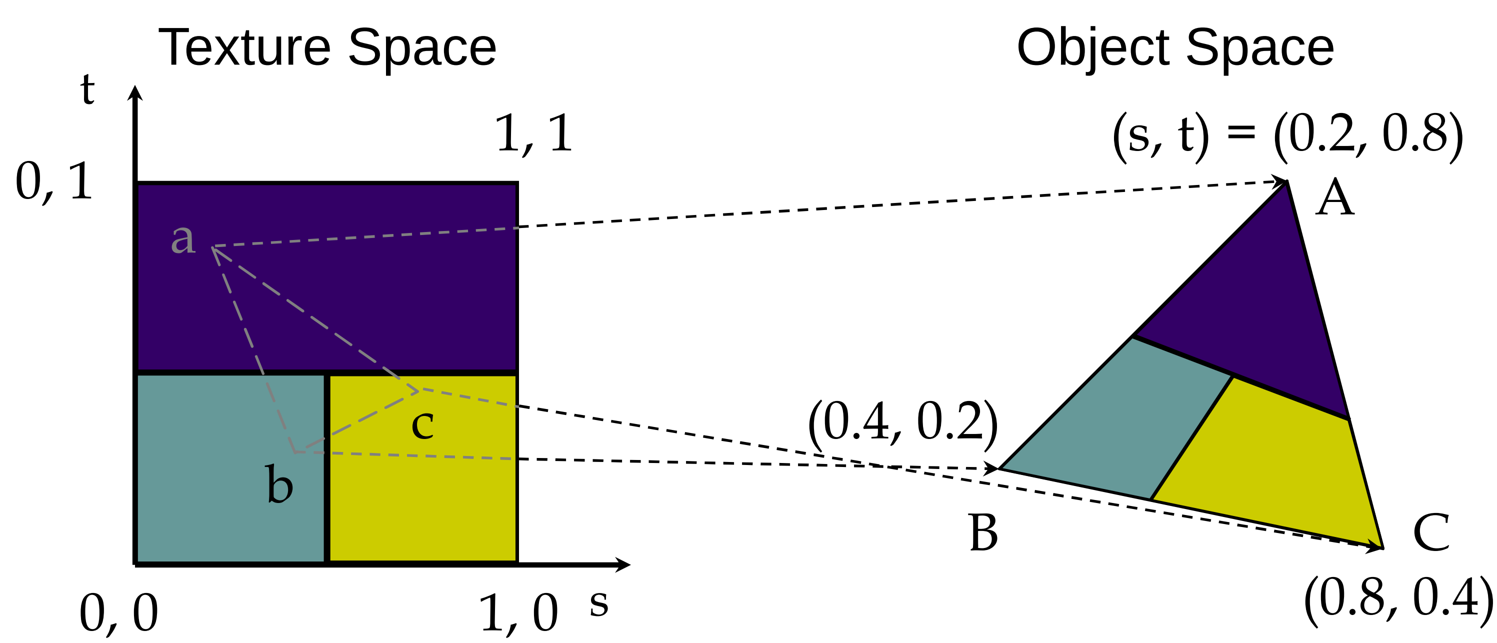
`texels:` pointer to texel array

E.g

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

Assigning Texture Coordinates

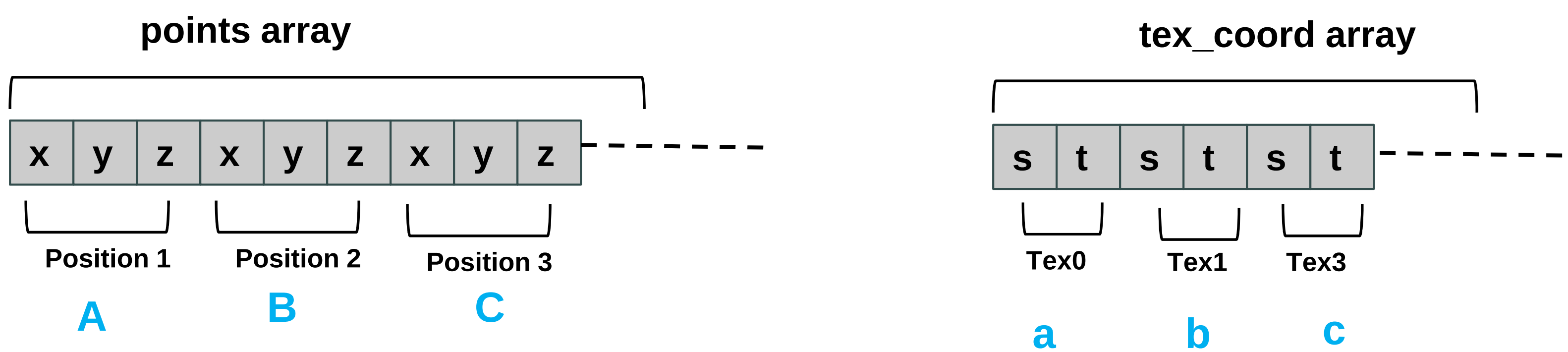
- Specify texture (s,t) coordinate each vertex (x,y,z) coordinate maps to
- E.g: (s,t) of ***a*** in texture => (x,y,z) of ***A*** on object
- Texture coordinates specified at each vertex



Code for Assigning Texture Coordinates

- Example: Trying to map a picture to a quad
- For each quad specify vertex (x,y,z), as well as texture coordinate it maps to in picture
- May generate array of vertices + array of texture coordinates

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



Adding Texture Coordinates

```
void quad( int a, int b, int c, int d )
{
    quad_colors[Index] = colors[a];
    points[Index] = vertices[a];
    tex_coords[Index] = vec2( 0.0, 0.0 );
    index++;
   quad_colors[Index] = colors[b];
    points[Index] = vertices[b];
    tex_coords[Index] = vec2( 0.0, 1.0 );
    Index++;

// other vertices
}
```

Passing Texture data

- 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 );
```

Vertex Shader

- Vertex shader receives data, output texture coordinates to be rasterized => to fragment shader
- Must do all other standard tasks too
  - Compute vertex position
  - Compute vertex color if needed

```
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
```

Applying Textures

- Textures are applied during fragments shading by a **sampler**
- Samplers return a texture color from a texture object

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

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

Output color Of fragment (gl_FragColor)

Original color Of fragment (color)

Lookup color of texCoord (s,t) in texture (texture2D(texture, texCoord))

Linking with Shaders

```
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. In this case,
//   zero, for GL_TEXTURE0 which was previously set by calling
//   glActiveTexture().
glUniform1i( glGetUniformLocation(program, "texture"), 0 );
```

Texture Parameters

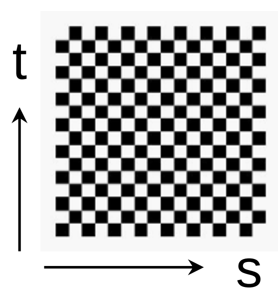
- OpenGL has a variety of parameters that determine how texture is applied
  - **Wrapping parameters** determine what happens if s and t are outside the (0,1) range
  - **Filter modes allow us to use area averaging** instead of point samples
  - **Mipmapping** allows us to use textures at multiple resolutions
  - **Environment** parameters determine how texture mapping interacts with shading

Wrapping Mode

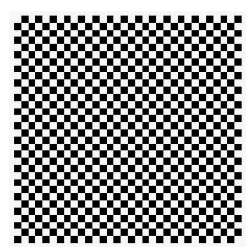
Clamping: if s,t > 1 use 1, if s,t <0 use 0

Wrapping: use s,t modulo 1

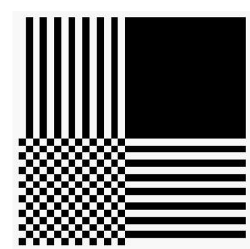
```
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL_REPEAT wrapping

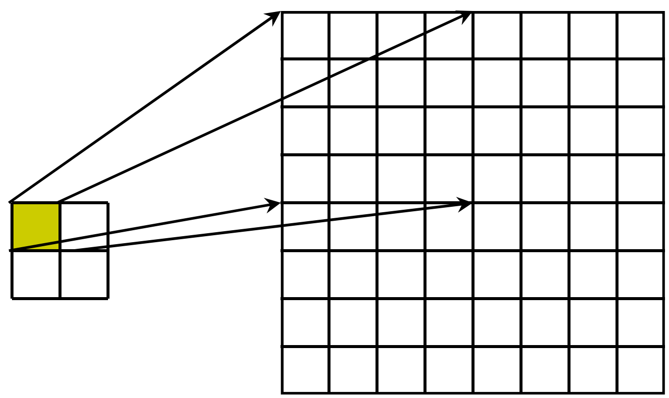


GL_CLAMP wrapping

Magnification and Minification

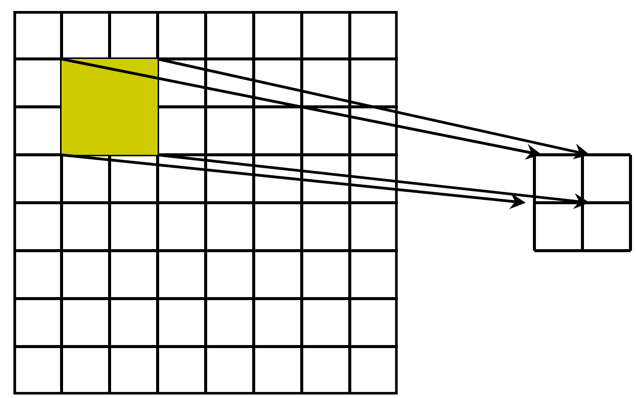
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Texture Polygon

Magnification



Texture Polygon

Minification

Filter Modes

Modes determined by

- **`glTexParameteri( target, type, mode )`**

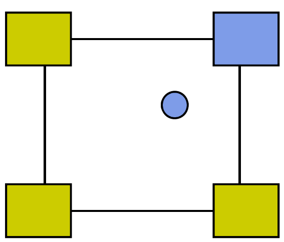
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
                GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_LINEAR);
```

Texture mapping parameters

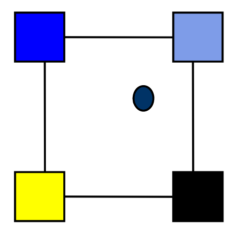
- OpenGL texture filtering:
- E.g. minification: pixel maps to more than 1 texel

1) Nearest Neighbor (lower image quality)



glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

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



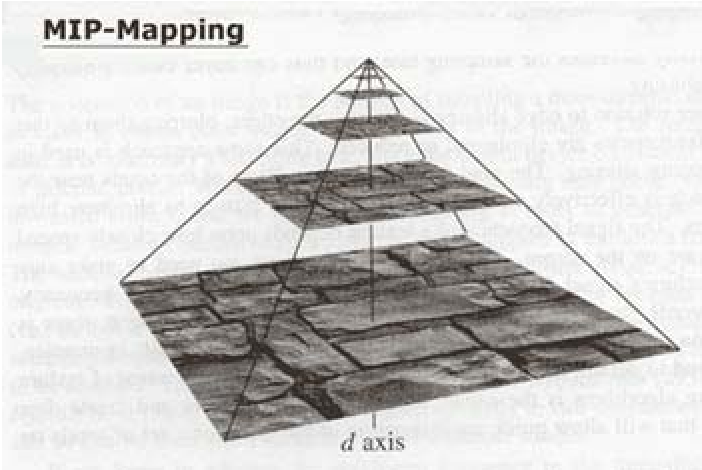
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

Or GL_TEXTURE_MAX_FILTER

Mipmapped Textures

- *Mipmapping* allows for prefiltered texture maps of decreasing resolutions
- Declare mipmap level during texture definition

```
glTexImage2D( GL_TEXTURE_*D, level, … )
```



Texture Functions

- Controls how texture is applied
  - **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);

Other Stuff

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

$$s = \frac{\theta - \theta_a}{\theta_b - \theta_a} \qquad t = \frac{z - z_a}{z_b - z_a}$$

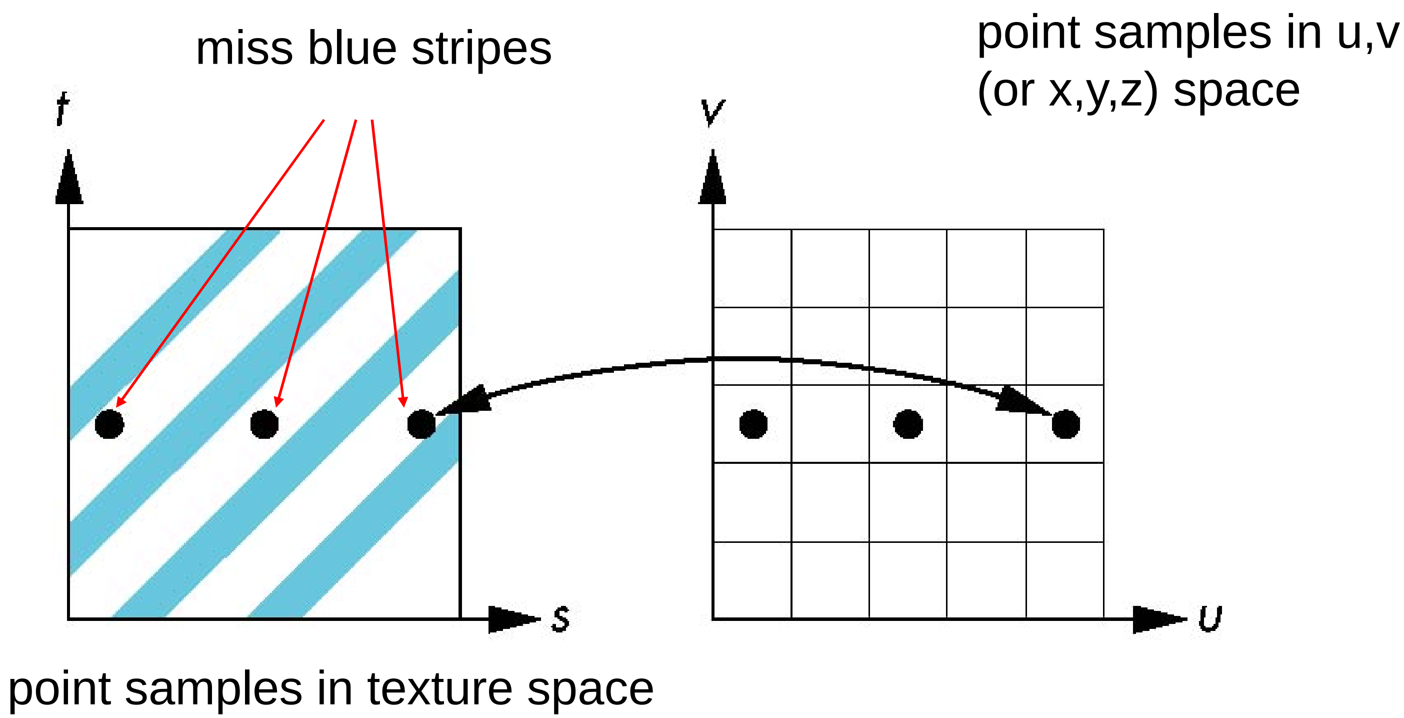
- Wrapping texture onto sphere

$$s = \frac{\theta - \theta_a}{\theta_b - \theta_a} \qquad s = \frac{\phi - \phi_a}{\phi_b - \phi_a}$$

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

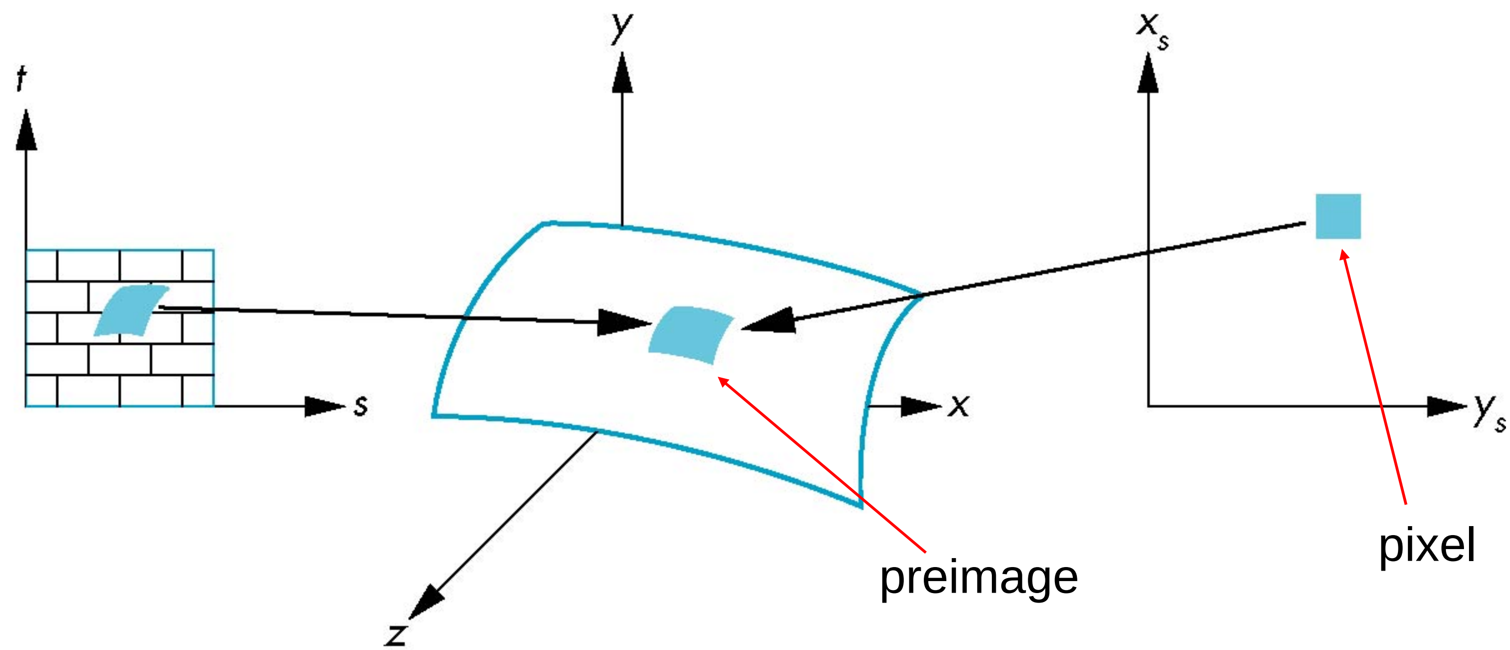
Aliasing

- Point sampling of the texture can lead to aliasing errors



Area Averaging

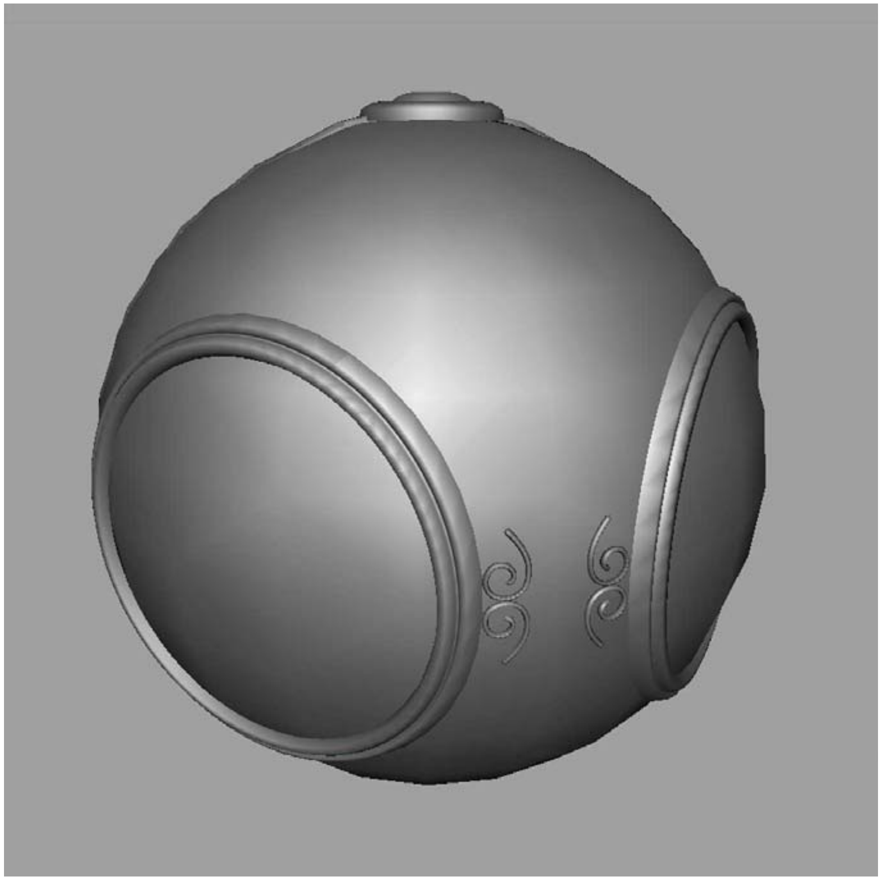
A better but slower option is to use *area averaging*



Note that *preimage* of pixel is curved

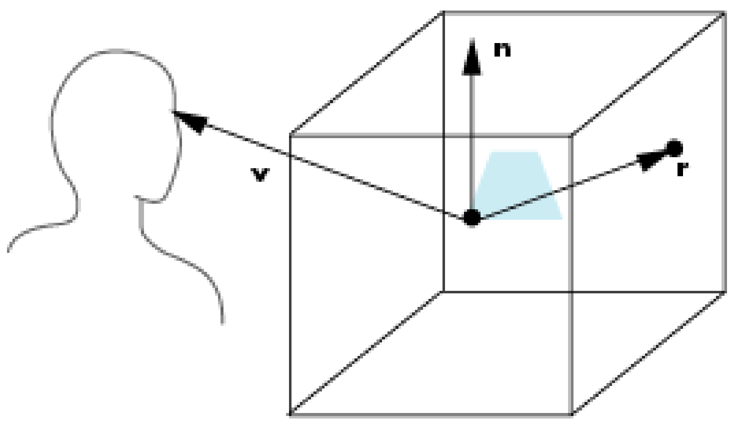
Environment Mapping

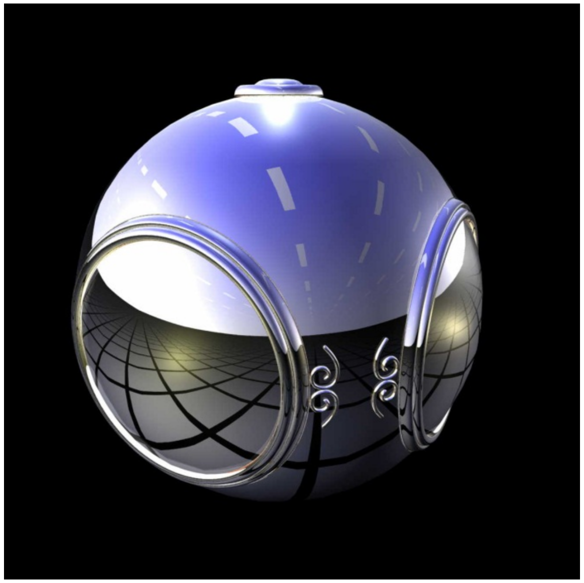
- Environmental mapping is way to create the appearance of highly reflective surfaces



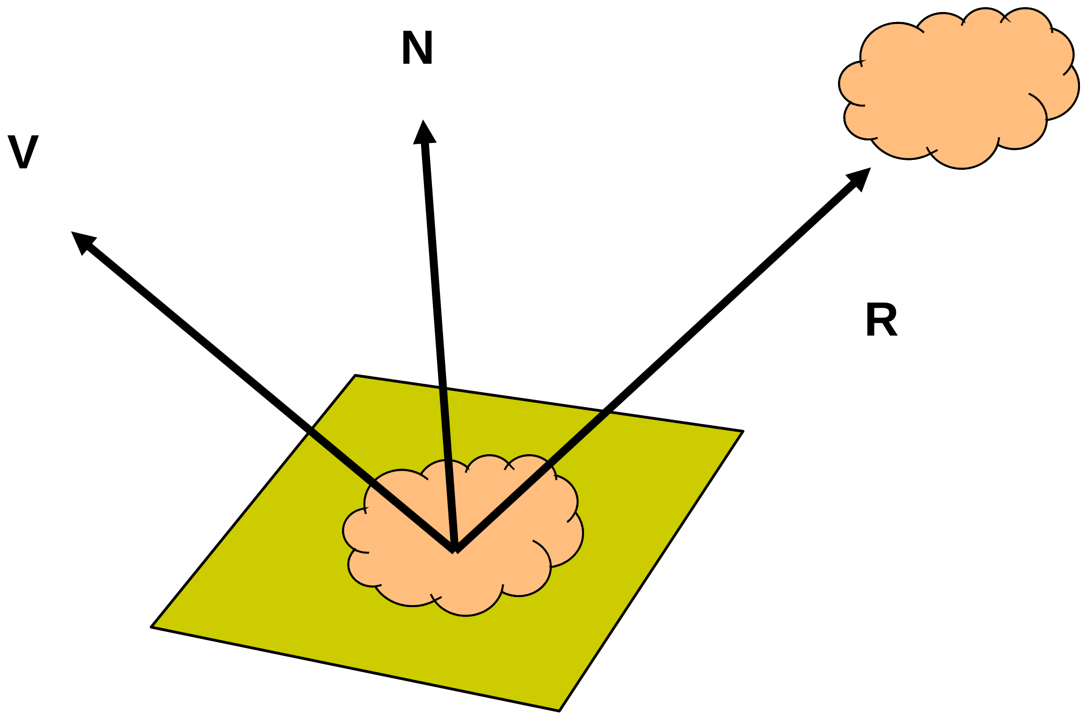
Environment Map

Use reflection vector to locate texture in cube map

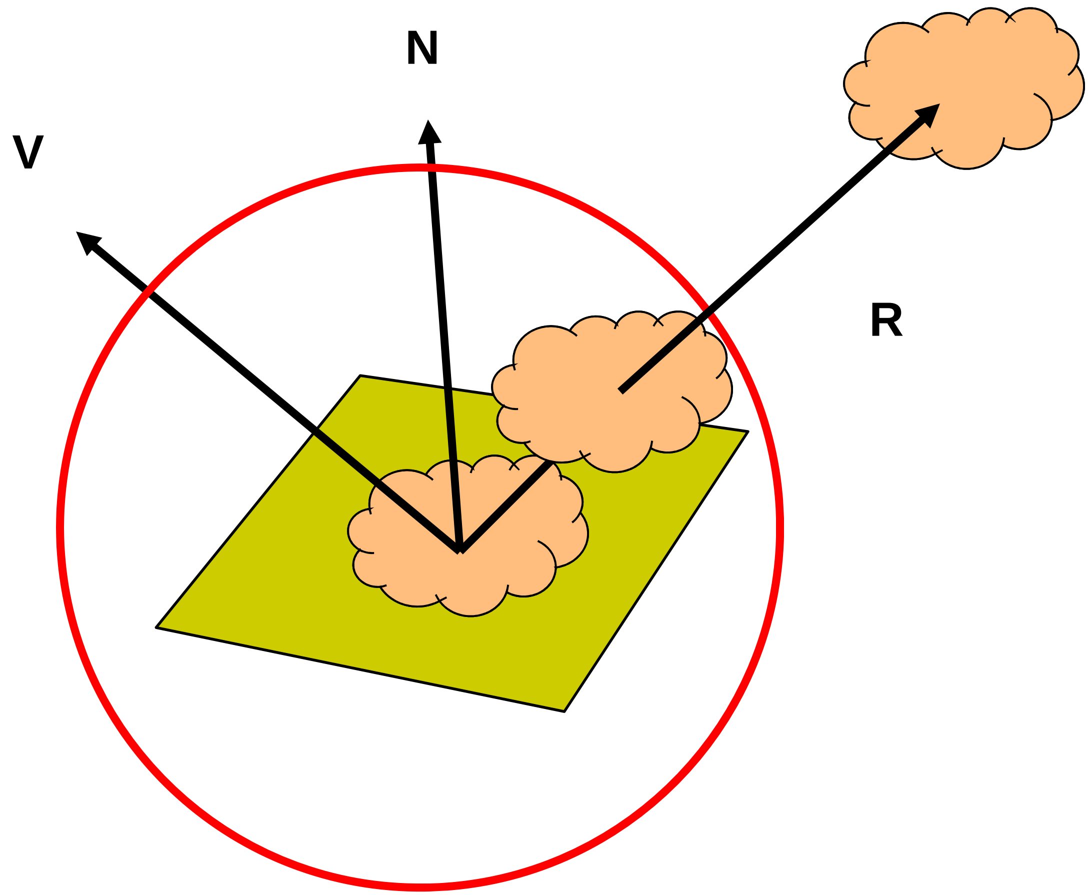




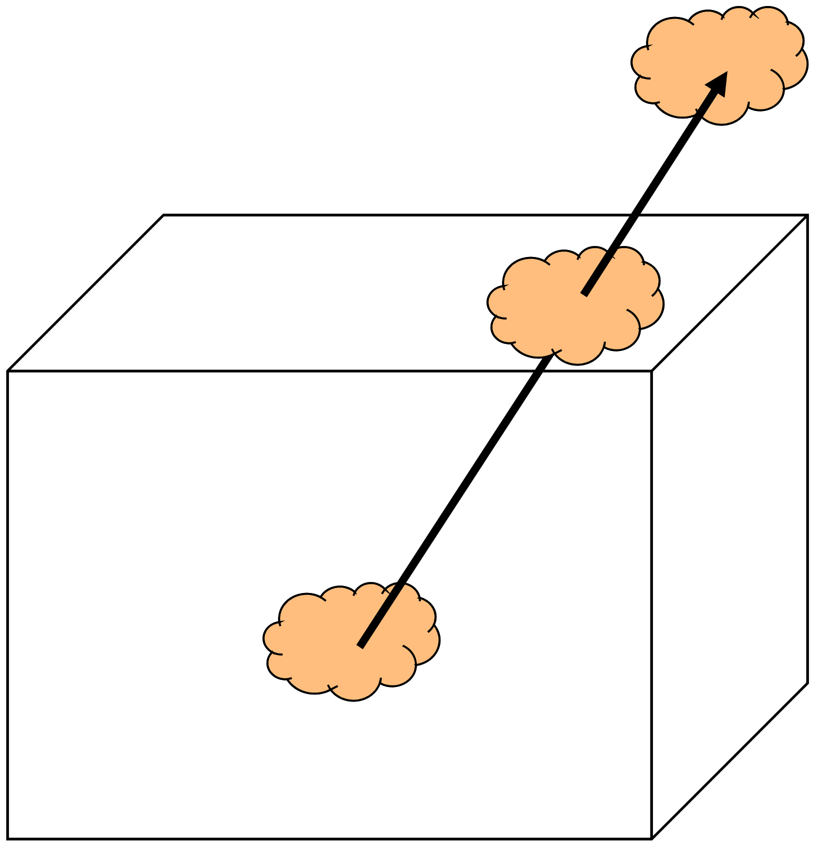
Reflecting the Environment



Mapping to a Sphere

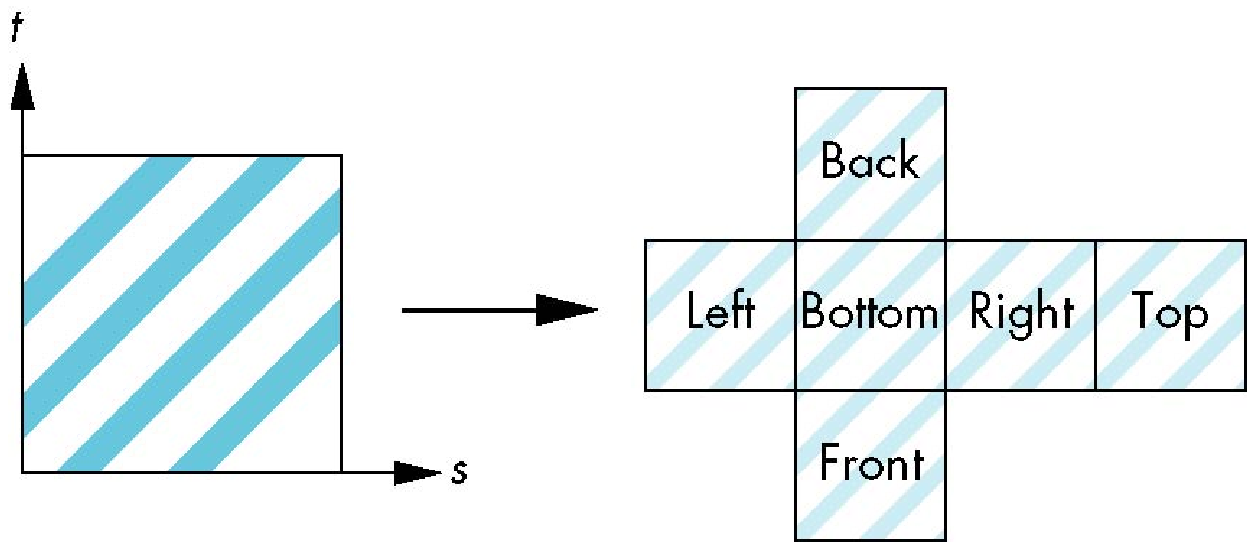


Cube Map



Box Mapping

- Map environment map to sides of a box



Cube Maps

- Cube map texture (as 6 sides of box) supported by OpenGL
- 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