Chapter 7.
Discrete Techniques:
Texture Mapping

Chapter 7.
Discrete Techniques:
Texture Mapping

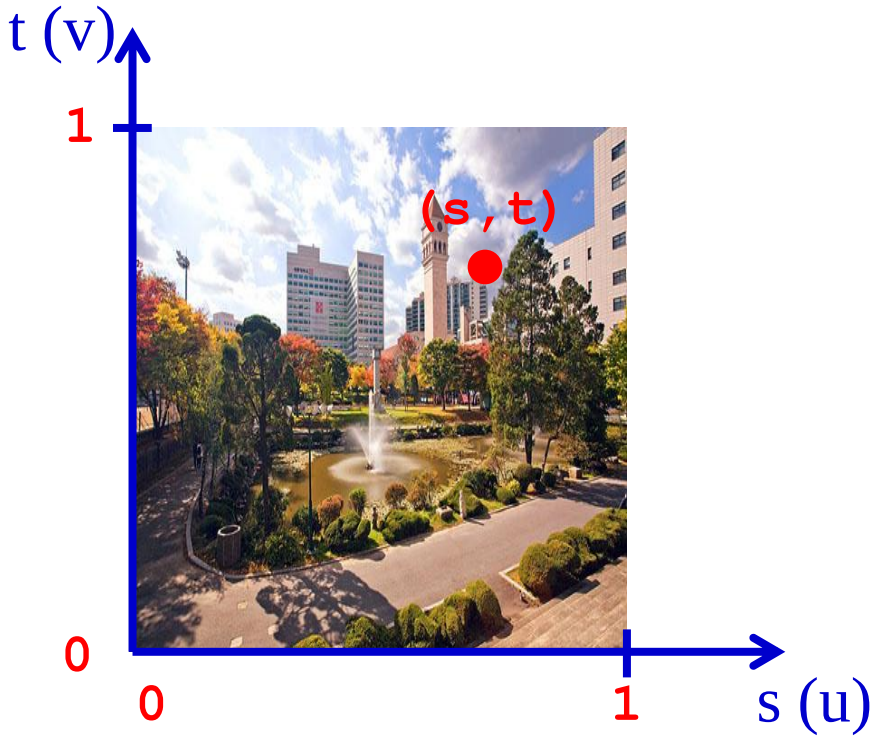
Map textures to surfaces

An image

Texture map

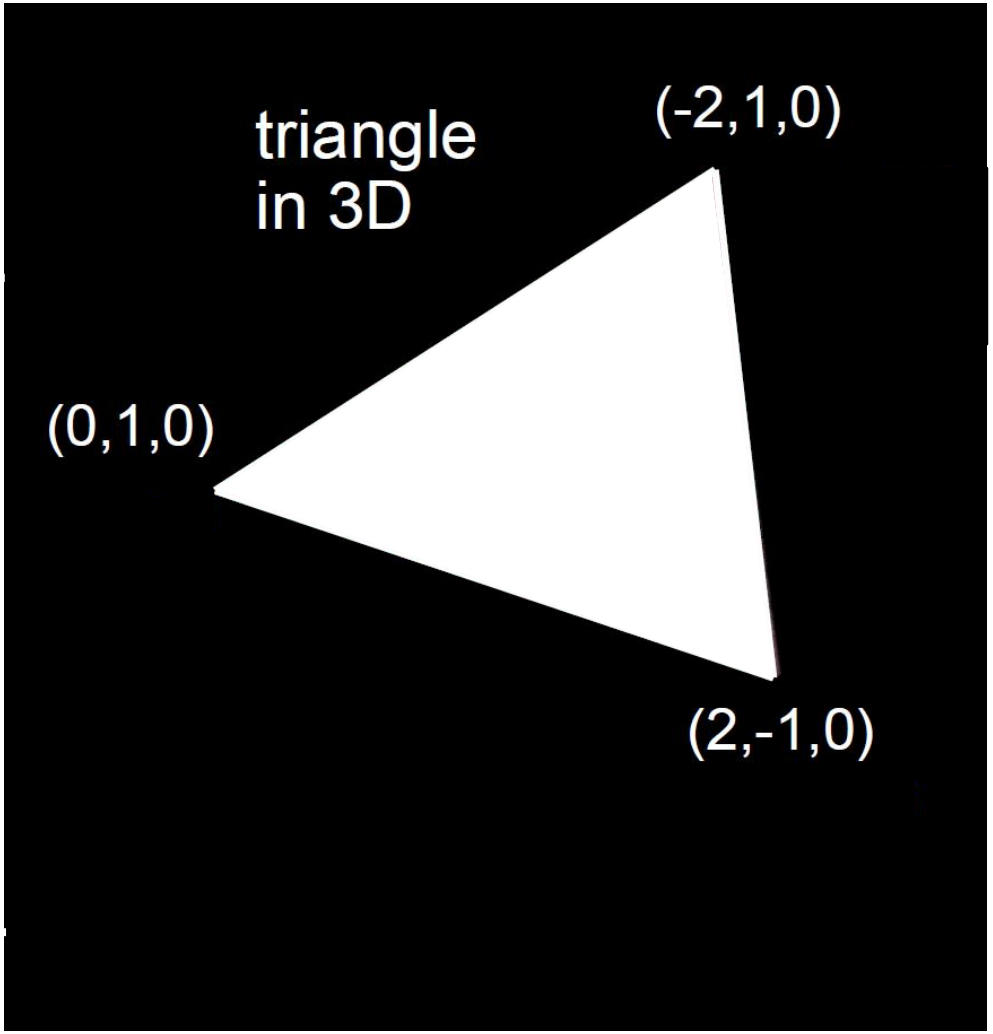
Image mapped to a 3D polygon:
The polygon can have arbitrary size, Shape, and 3D position.

The “st” coordinate system

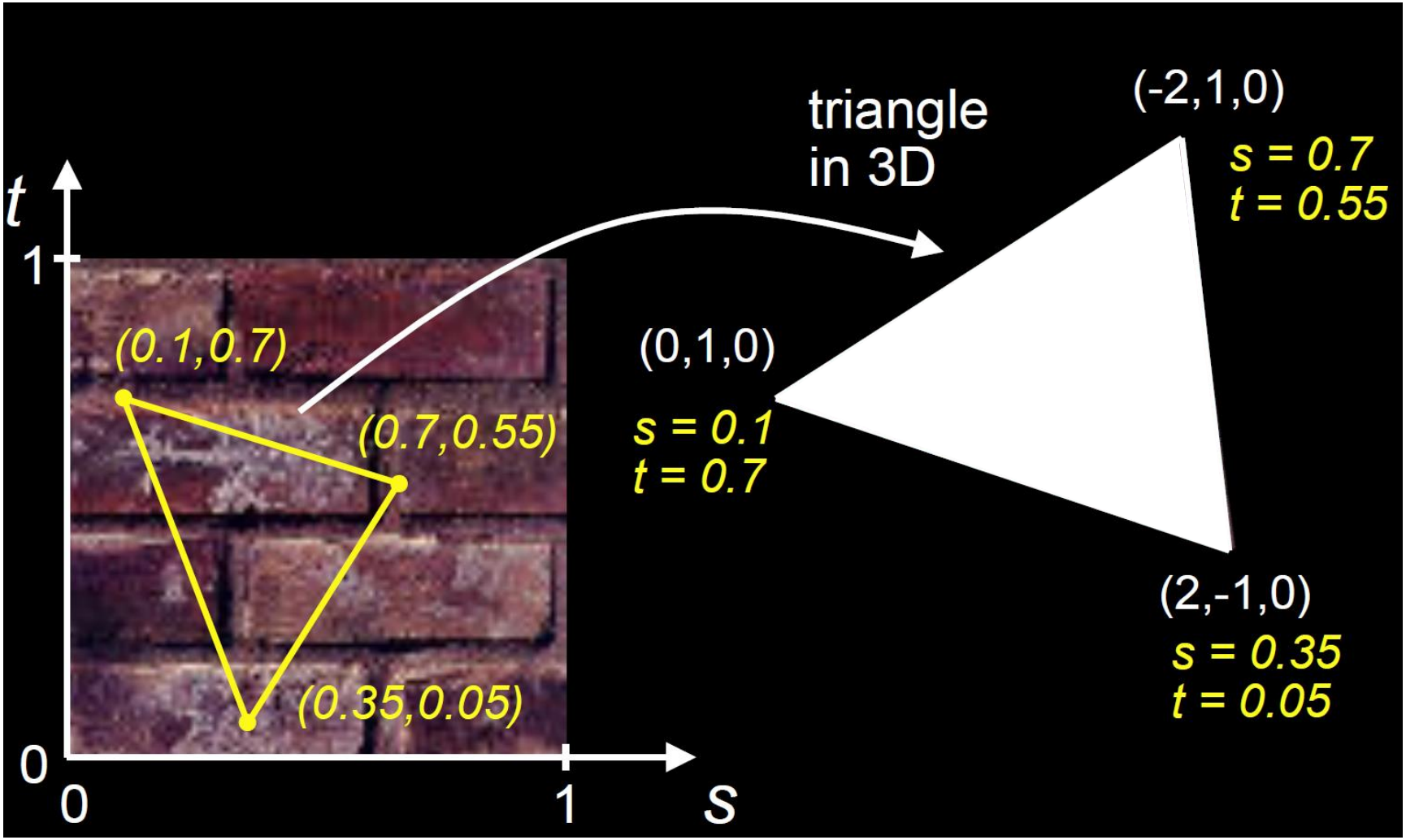


Note: also called “uv” space

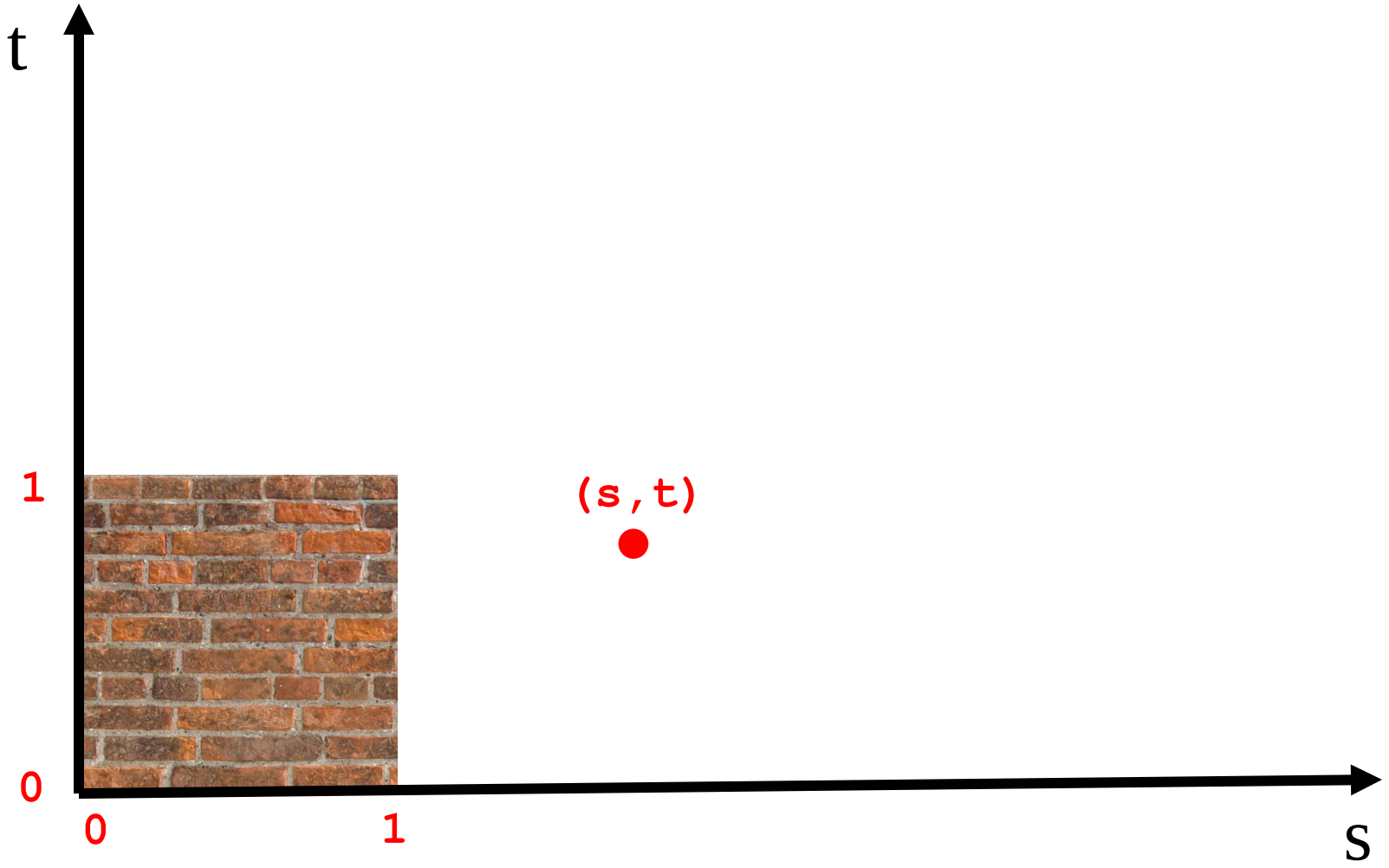
Texture mapping: key slide



Texture mapping: key slide

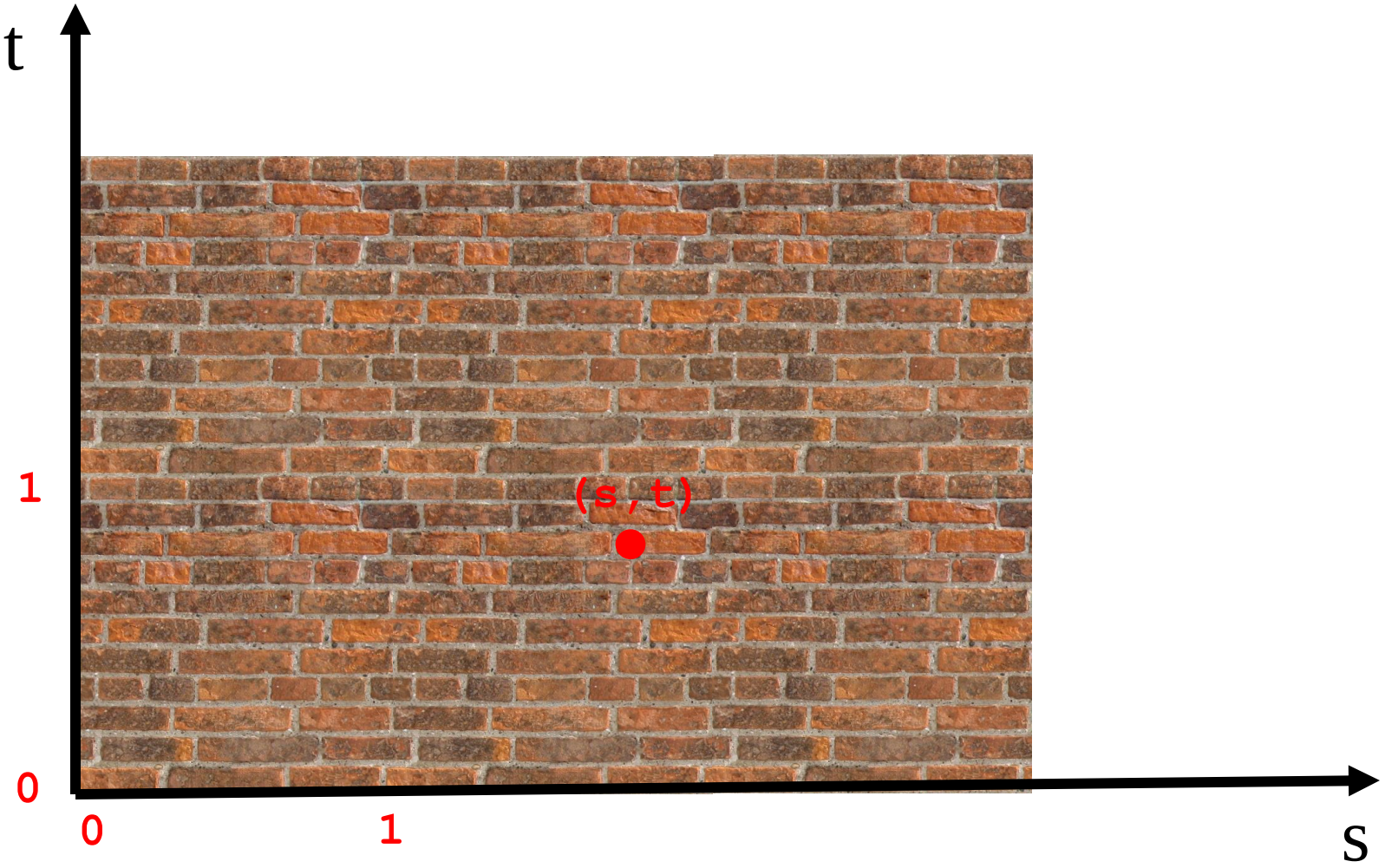


What if texture coordinates are outside of [0,1]



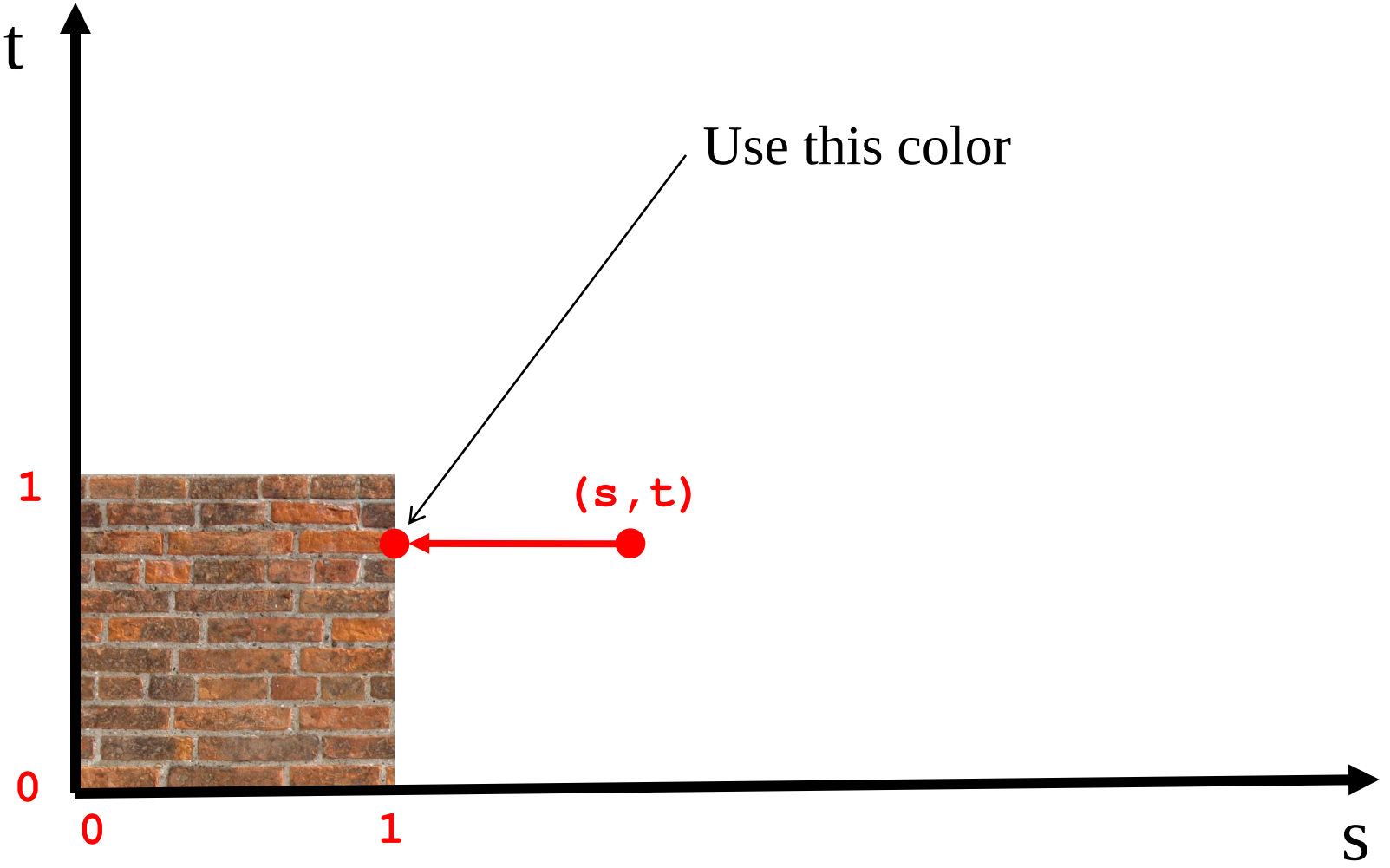
Solution 1: Repeat texture

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



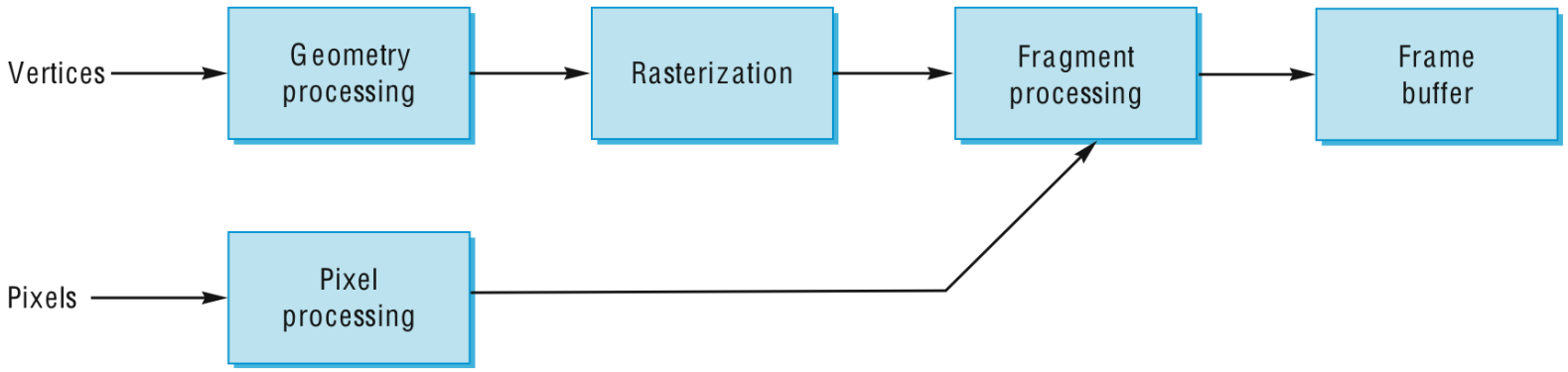
Solution 2: Clamp to [0,1]

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



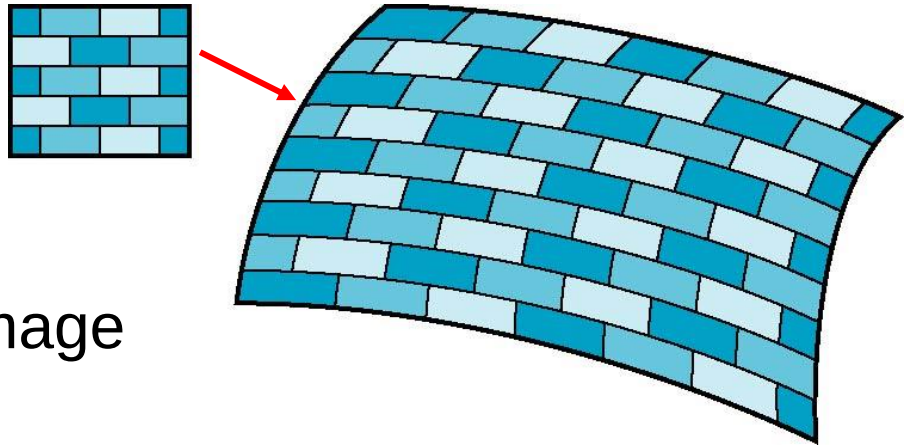
Where does mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline



How to compute the map?

- Is it simple?



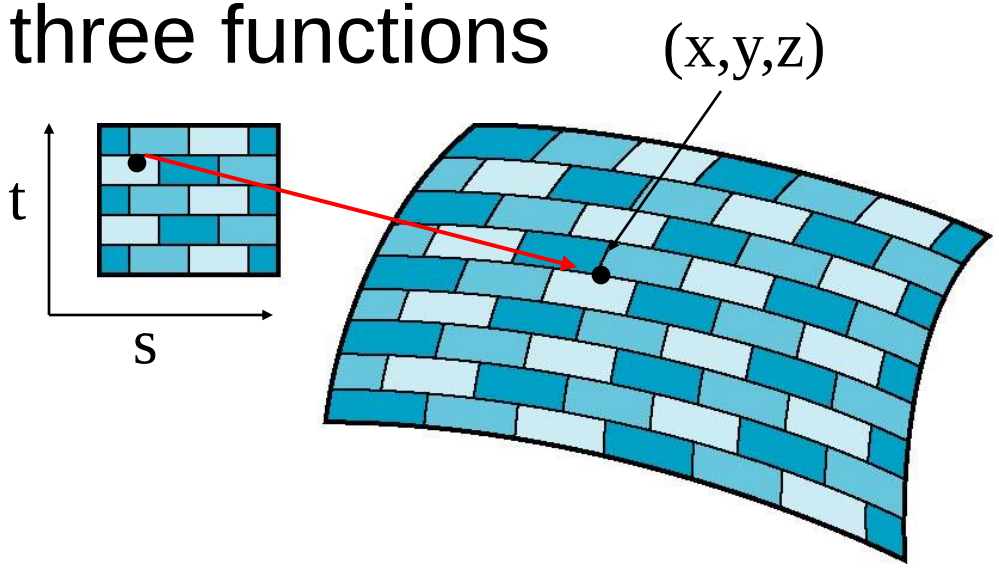
2D image

3D surface

Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

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



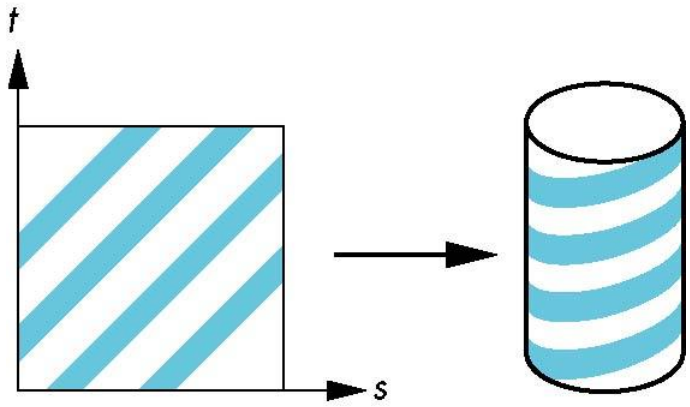
- But we really want to go the other way

Backward Mapping

- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form
  - $s = s(x,y,z)$
  - $t = t(x,y,z)$
- Such functions are difficult to find in general

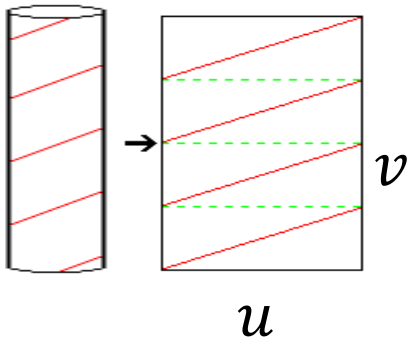
Two-part mapping

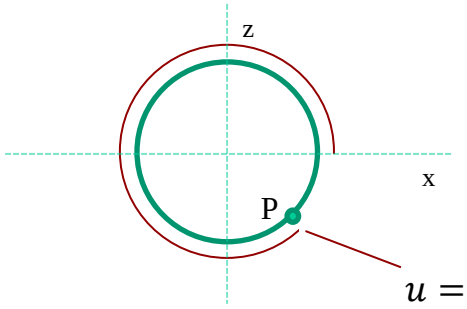
- One solution to the mapping problem is to first map the texture to a simple intermediate surface
- Example: map to cylinder

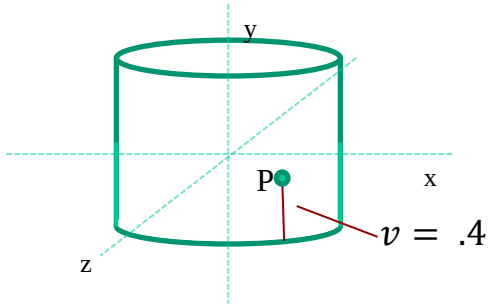


Cylindrical Mapping

- How to texture map cylinders and cones:
- Given a point P on the surface:
  - If it's on one of the caps, map as though the cap is a plane
  - If it's on the curved surface:
    - Use the position of the point around the perimeter to determine u
    - Use the height of the point to determine v

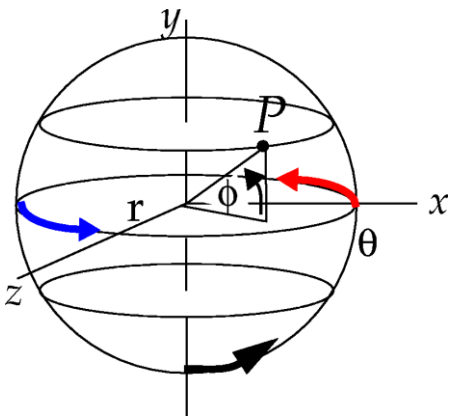






Spherical Map

- Texture mapping spheres:
  - Find (u, v) coordinates for P
  - We compute u the same we do for cylinders and cones
  - If $v = 0$ or $v = 1$, there is a singularity. Set u to some predefined value. (0.5 is good)
  - v is a function of the latitude of P



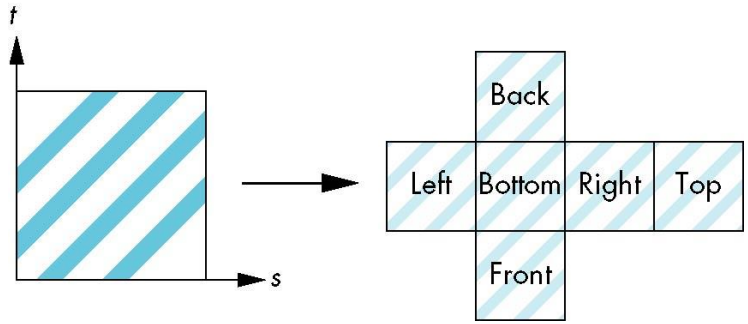
$$\phi = \sin^{-1}\frac{P_y}{r} \qquad -\frac{\pi}{2} \le \phi < \frac{\pi}{2}$$

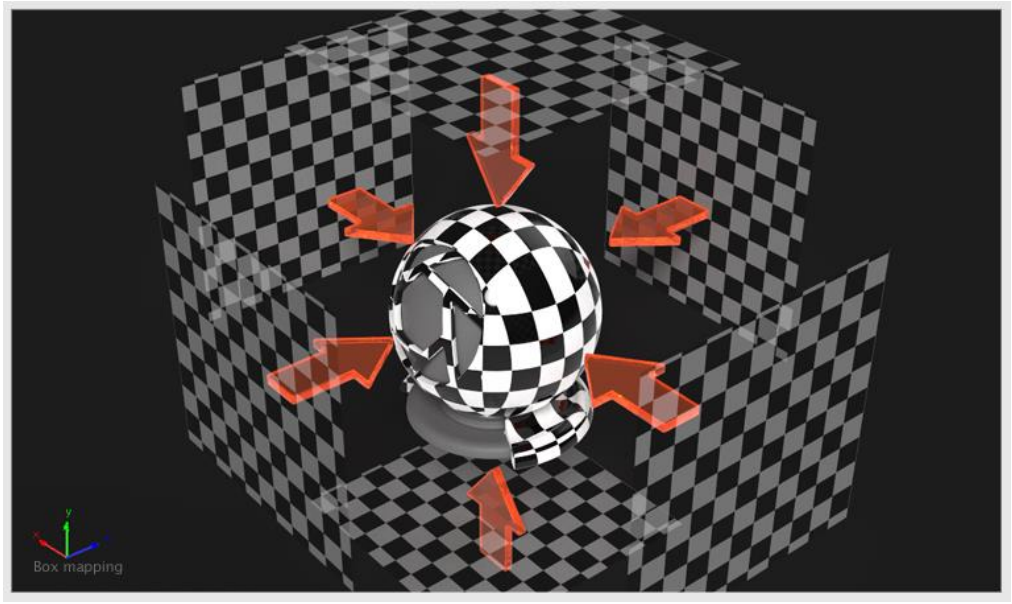
r = radius

$$v = \frac{\phi}{\pi} + \frac{1}{2}$$

Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps





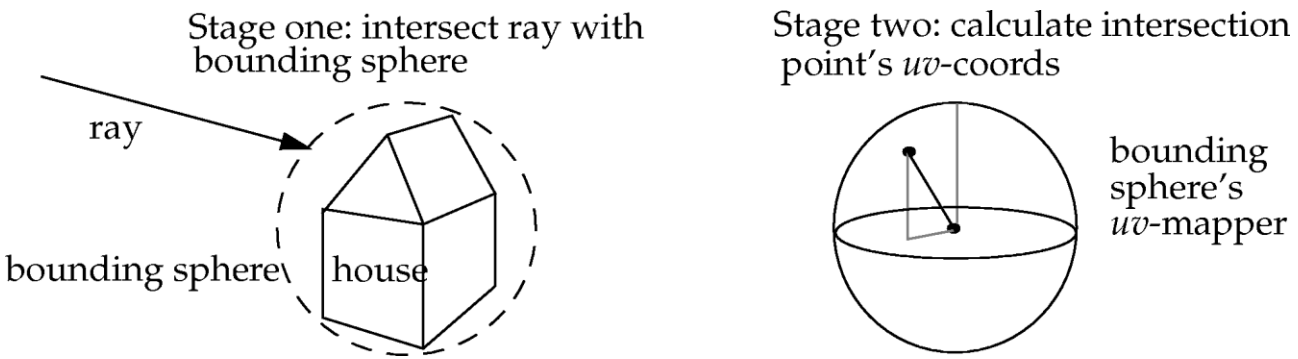
Texture Mapping Complex Geometry

- Sometimes, reducing objects to primitives for texture mapping doesn't achieve the right result.
  - Consider a simple house shape as an example
  - If we texture map it using polygons, we get discontinuities at some edges.

- Solution: Pretend object is a sphere and texture map using the sphere (u, v) map

Texture Mapping Complex Geometry

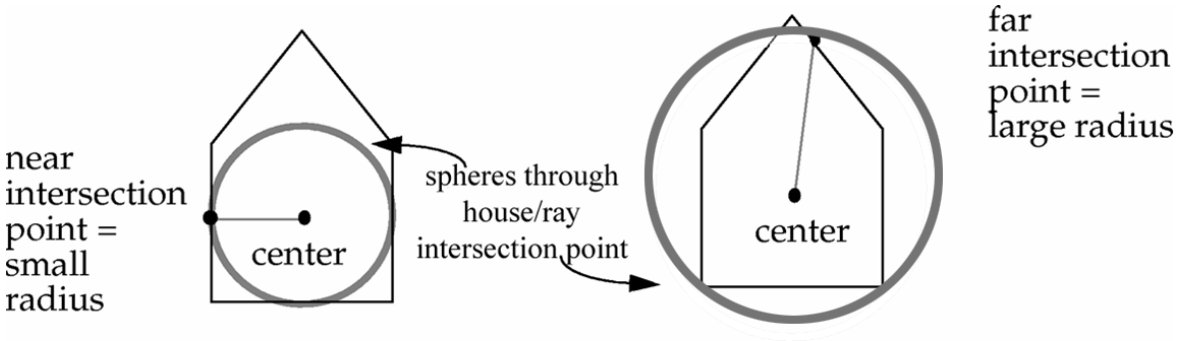
- Intuitive approach: Place a bounding sphere around the complex object
  - Find ray's object space intersection with bounding sphere
  - Convert to (u, v) coordinates



- We actually don't need a bounding sphere!
  - Once we have the intersection point with the object, we just treat it as though it were on a sphere passing through the point. Same results, but different radii.

Texture Mapping Complex Geometry

- When we treat the object intersection point as a point on a sphere passing through the point, our “sphere” won’t always have the same radius



- What radius to use?
  - Compute radius as distance from defined or computed center of complex object to the intersection point. Use that as the radius for the (u, v) mapping.

Texture Mapping Complex Geometry

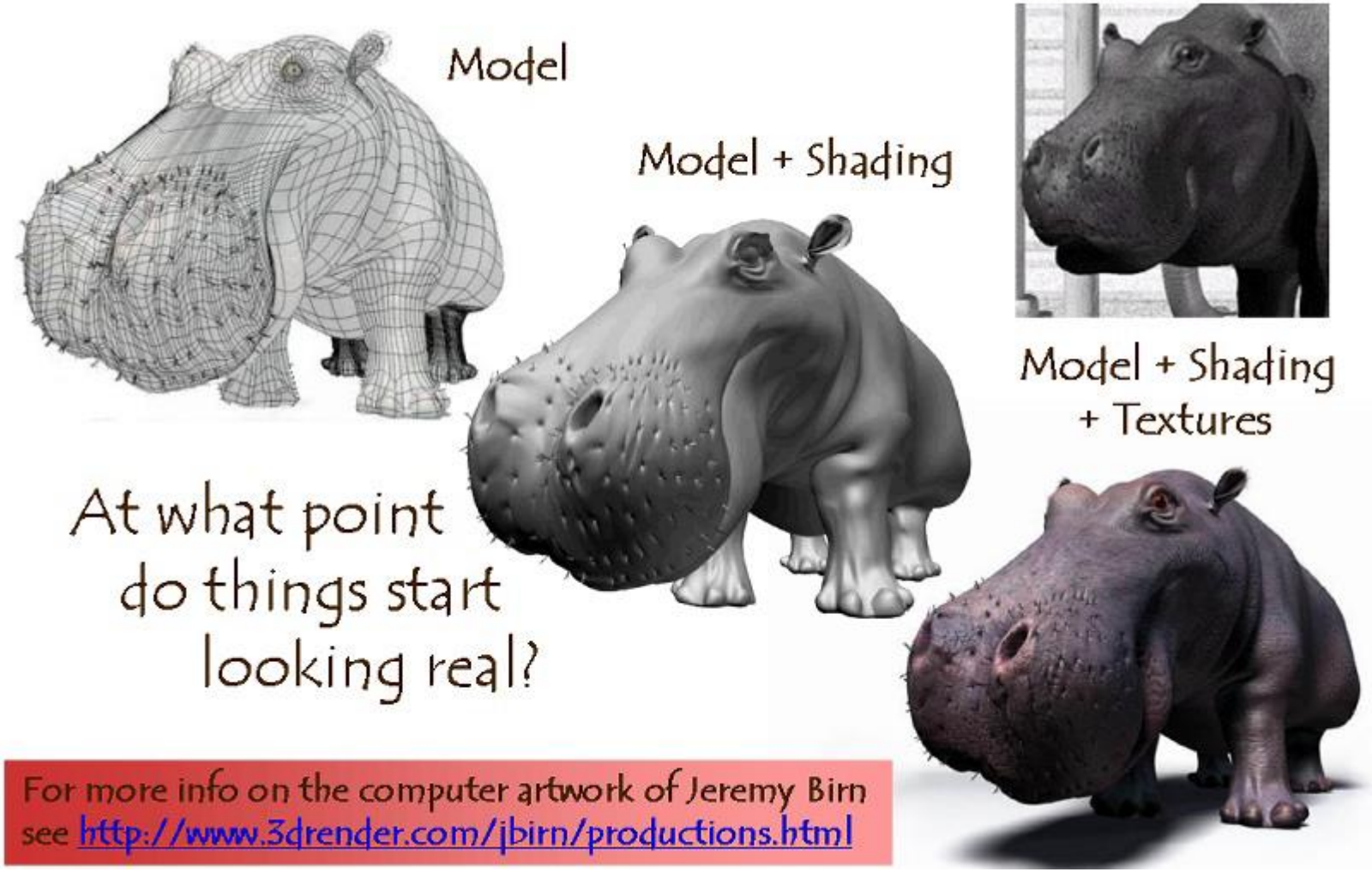
- Results of spherical (u, v) mapping on house:

- You can also use cylindrical or planar mappings when texture mapping complex objects
  - Each has drawbacks
    - Spherical: warping at the “poles” of the object (the top of the house)
    - Cylindrical (not shown here): discontinuities at the caps
    - Planar: one axis is ignored

sphere mapped with spherical projection

sphere mapped with planar projection

Combining texture mapping and shading



Combining texture mapping and shading

- Final pixel color = a combination of texture color and color under standard OpenGL Phong lighting

- GL_MODULATE:
  multiply texture and Phong lighting color

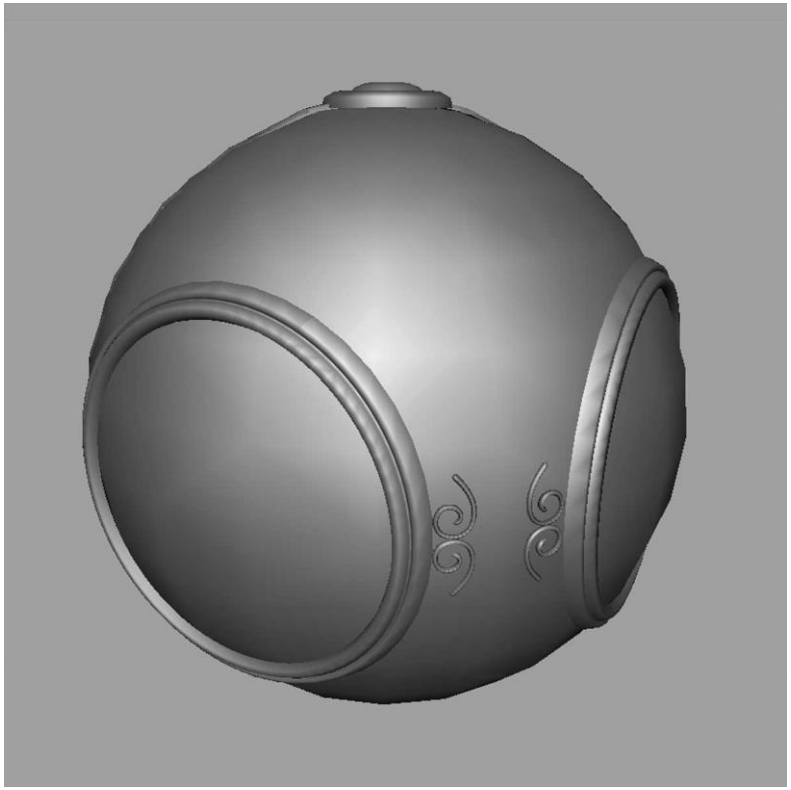
- GL_BLEND:
  linear combination of texture and Phong lighting color

- GL_REPLACE:
  use texture color only (ignore Phong lighting)

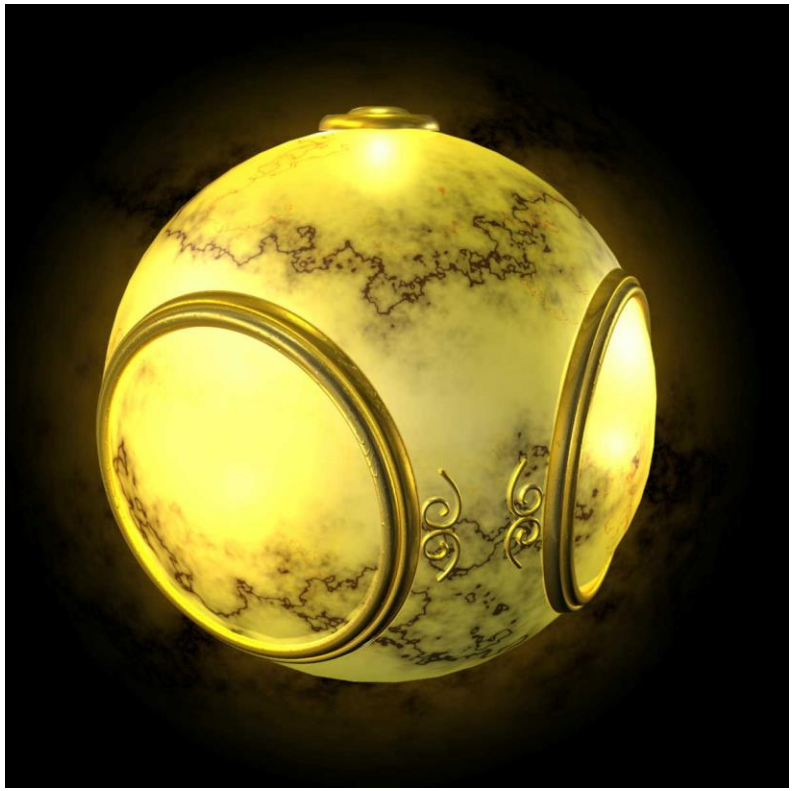
- Example:

```
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

Texture Mapping



geometric model



texture mapped

OpenGL Texture Mapping

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

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
  - Scanned image
  - Generate by application code

Creating a Texture Object

```
GLuint textures;
glGenTextures( 1, &textures );
glActiveTexture( GL_TEXTURE0 );

glBindTexture( GL_TEXTURE_2D, textures );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, Width, Height,
                  0, GL_RGB, GL_UNSIGNED_BYTE, image );
```

Sending a Texture to GPU

```
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)

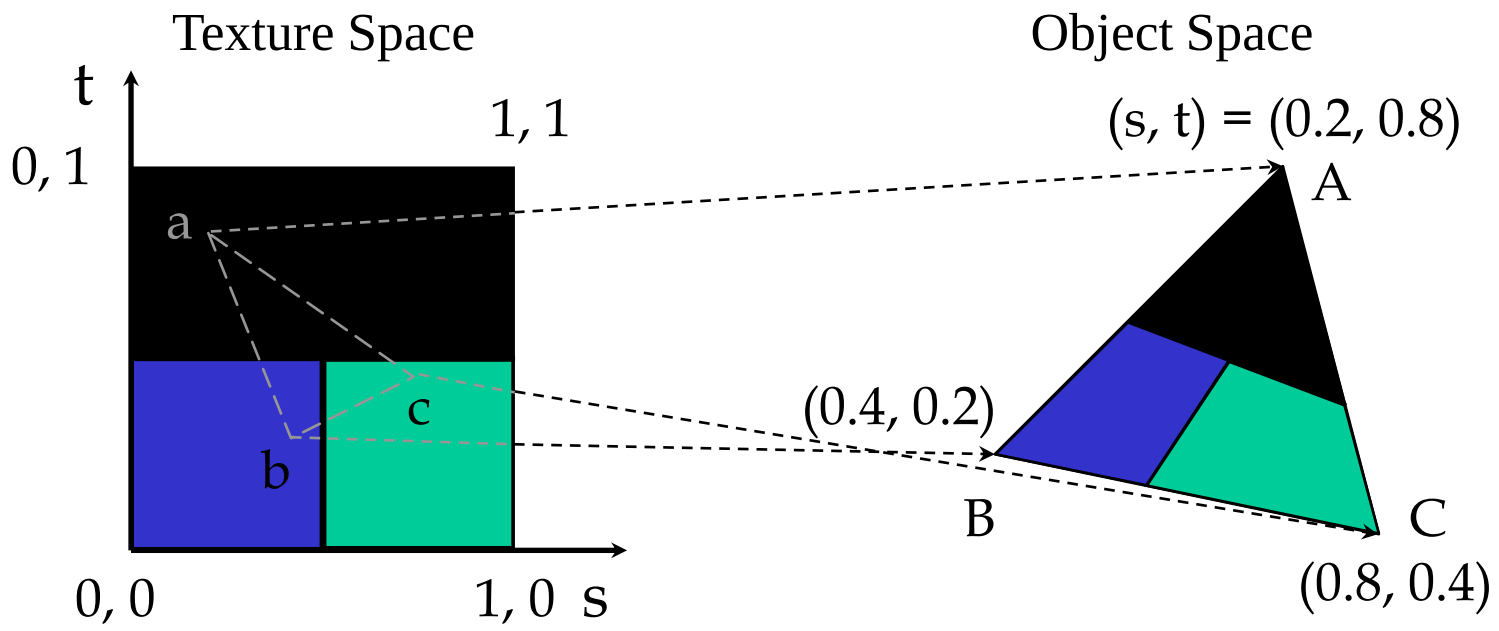
format and type: describe texels

texels: pointer to texel array

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

Mapping a Texture

- Based on parametric texture coordinates



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

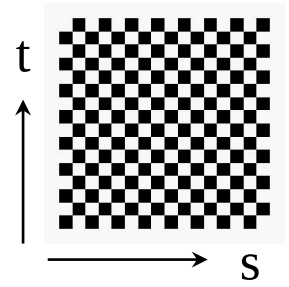
Parameter 1: Wrapping Mode

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

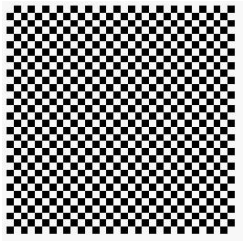
```
glTexParameteri( GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_S, GL_CLAMP )
```

Wrapping: use s,t modulo 1

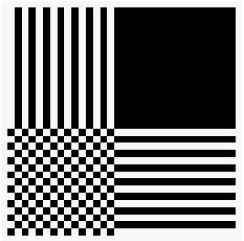
```
glTexParameteri( GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL_REPEAT wrapping

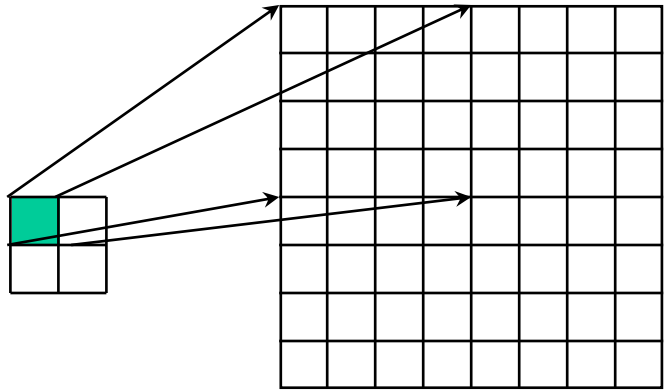


GL_CLAMP wrapping

Parameter 2: Magnification and Minification

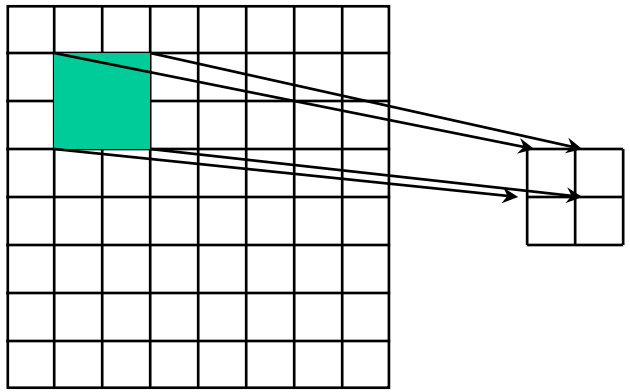
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

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



Texture Polygon

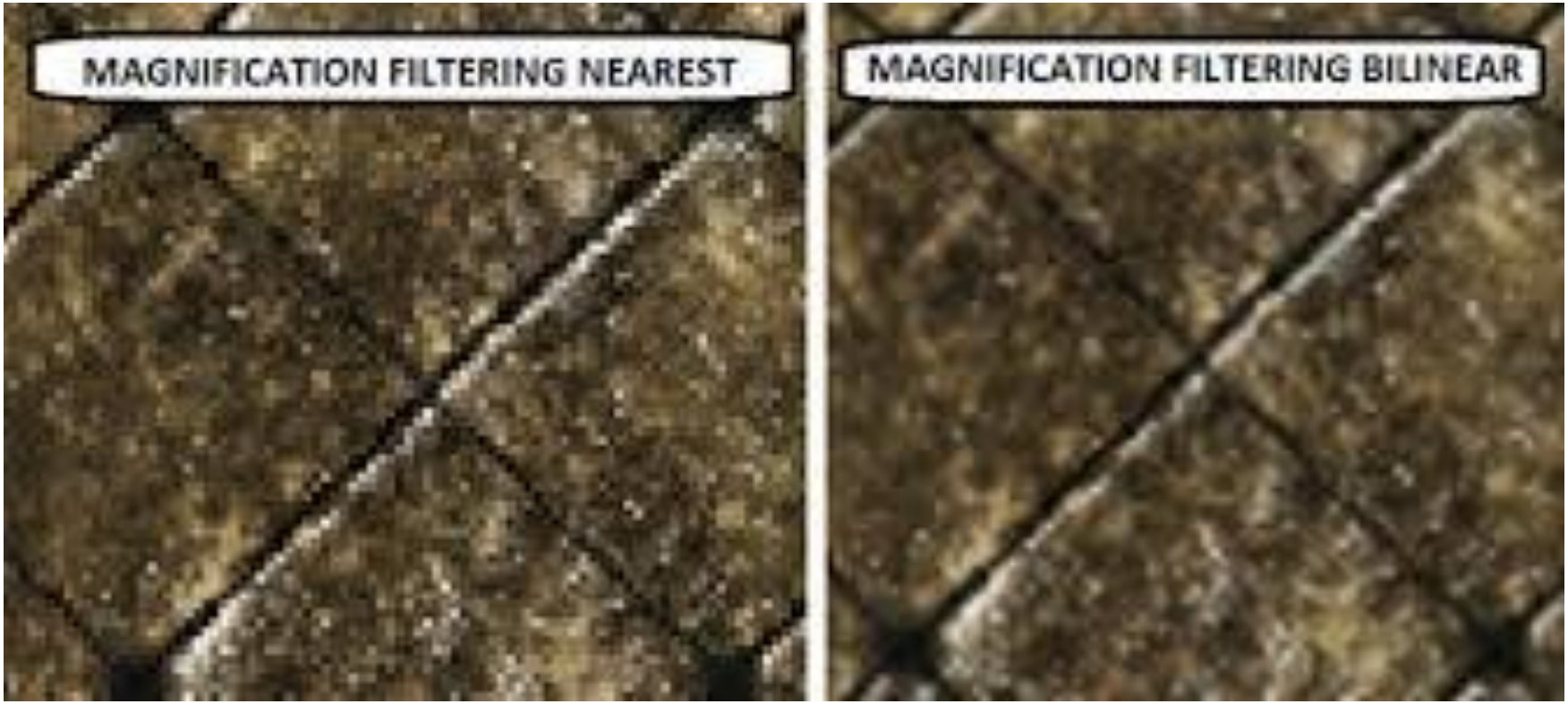
Magnification



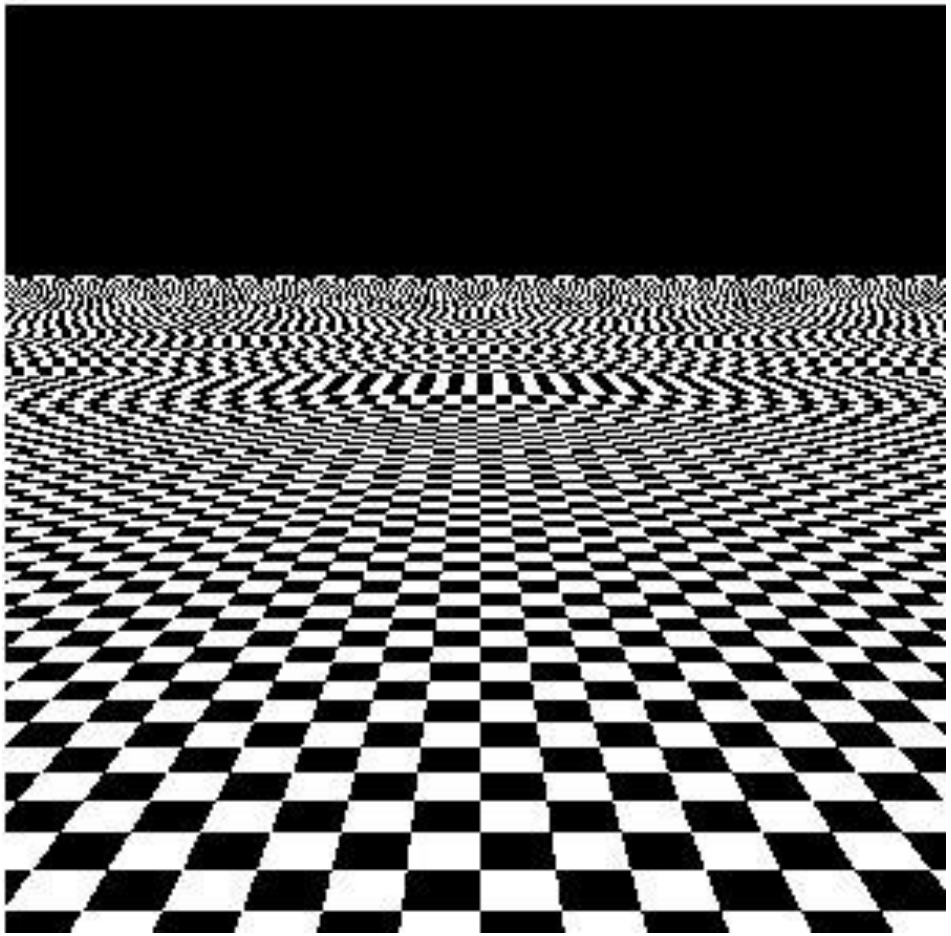
Texture Polygon

Minification

Texture Filtering in Magnification



Texture Aliasing Problem in Minification



Parameter2: Magnification and Minification

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);
```

Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1)

Summary of Creating Texture Object

```
GLuint textures;
glGenTextures( 1, &textures );
glActiveTexture( GL_TEXTURE0 );

glBindTexture( GL_TEXTURE_2D, textures );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, Width, Height,
              0, GL_RGB, GL_UNSIGNED_BYTE, image );

glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                 GL_REPEAT );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                 GL_REPEAT );

glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                 GL_NEAREST);
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST);
```

Using Texture Objects

1. specify textures in texture objects
2. bind texture object
3. set texture filter
4. set texture wrap mode
5. enable texturing
6. supply texture coordinates for vertex
   - coordinates can also be generated

Vertex Shader

- Usually vertex shader will output texture coordinates to be rasterized
- 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
```

Applying Textures in fragment shader

- 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 uTexture; //texture object from application

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

Linking the uniform 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 );
```

Using an Image as a texture

- An image can be used as a texture
- OpenGL does not support loading image.
- Use other library to load image:
  ex) OpenCV, QT (qt-project.org)

- Image should be converted to Unsigned Byte data array.

STGA class

- A Simple class for loading TGA image
- TGA image is the simplest image format.
- Download “targa.h” in our homepage

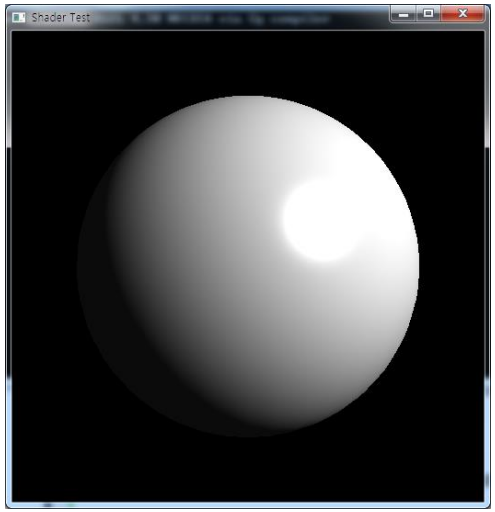
```
STGA image;

image.loadTGA("c:\\earthmap_dif.tga");}

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             image.width, image.height, 0, GL_BGR,
             GL_UNSIGNED_BYTE, image.data);
```

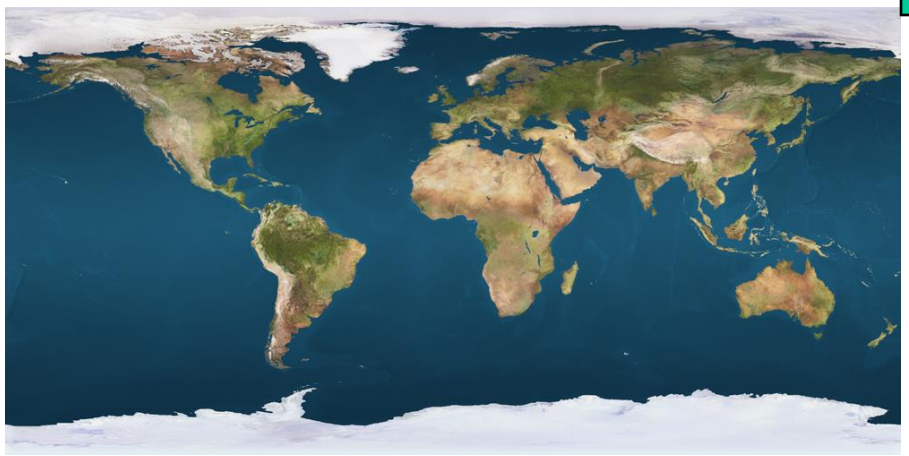
Example Code for reading a texture

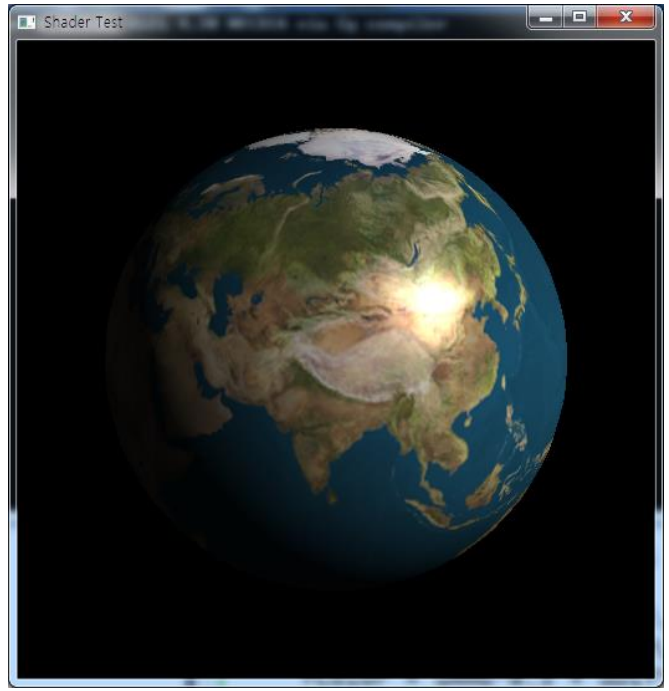
```
GLuint myTex;
glGenTextures(1, &myTex);

STGA img;                          // requires #include “targa.h”
img.loadTGA(“texture.tga”);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, myTex);

glTexImage2D(GL_TEXTURE_2D, 0, 3, img.width, img.height, 0, GL_BGR,
GL_UNSIGNED_BYTE, img.data);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
img.destroy();
```

```
// fragment shader:

uniform sampler2D uTex;                  // glUniform1i(uTex, 0);
color = texture2D(uTex, texCoord);
```

Coding practice: planet earth

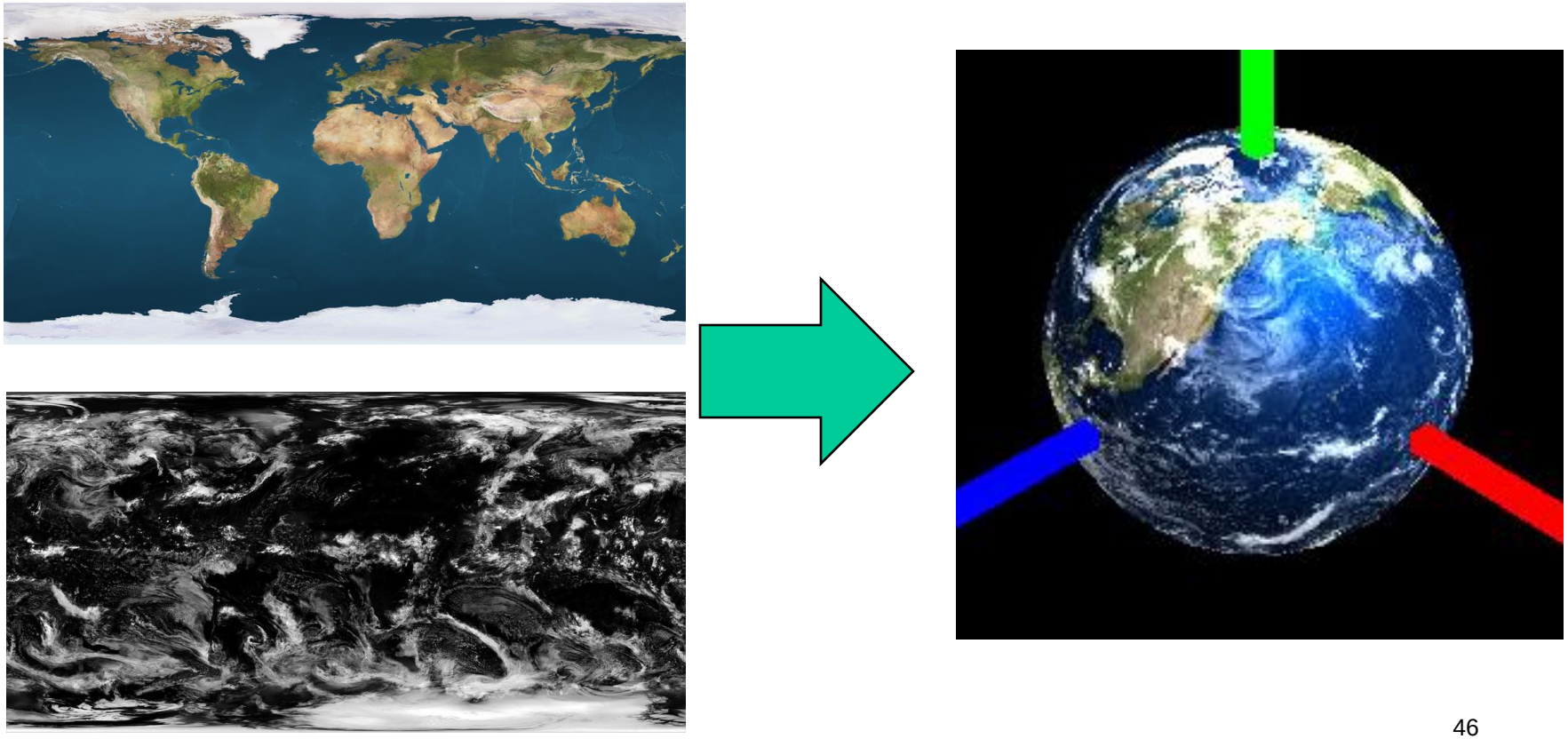






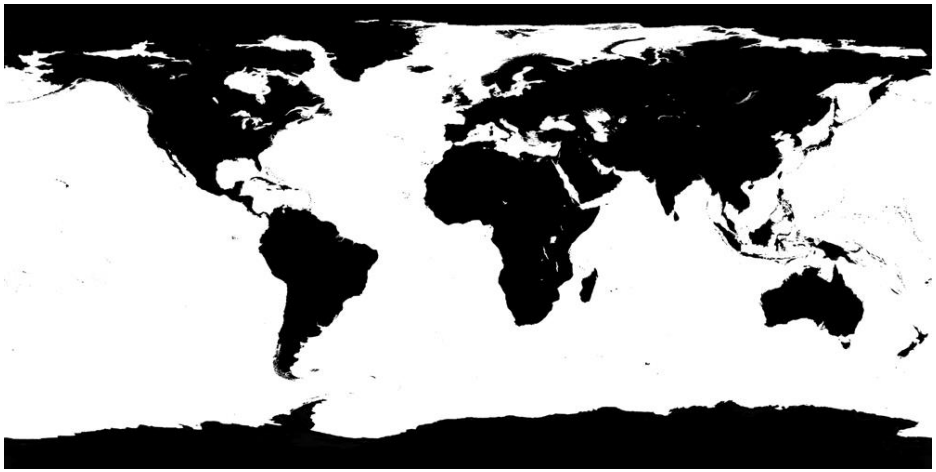
Multi-Texturing

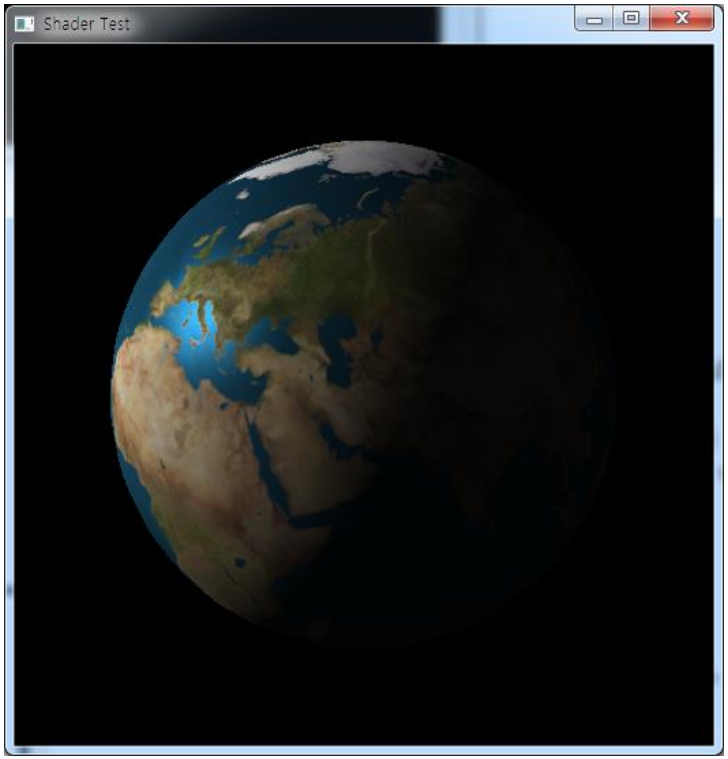
- Apply multiple textures at the same surface:



Multi-texturing: Specular Map:

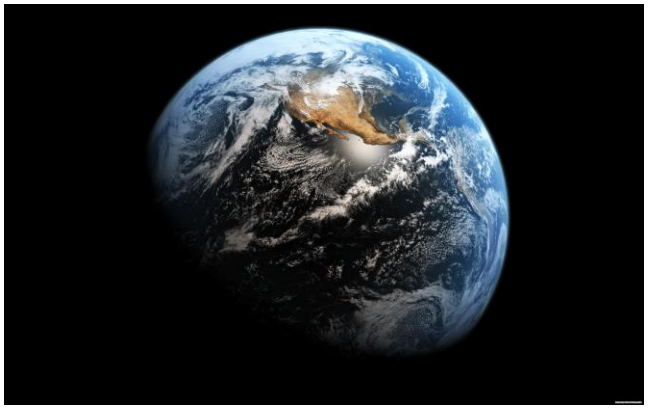
- Specify the specular reflectance by giving the additional texture (specular map)





Think more:

- Observe the photos of the earth
- What effects you need more and how to do implement?



Make your own Planet Earth: