Chapter 5.
Lighting and Shading

Shading Implementation with GLSL

Review: OpenGL shading

- Need
  - Normals
  - material properties
  - Lights
- State-based shading functions have been deprecated (glNormal, glMaterial, glLight)
- send attributes or uniforms to shaders

Review: Polygon Rendering Methods

- Curved surfaces are often approximated by polygonal surfaces

- So, polygonal (piecewise planar) surfaces often need to be rendered as if they are smooth

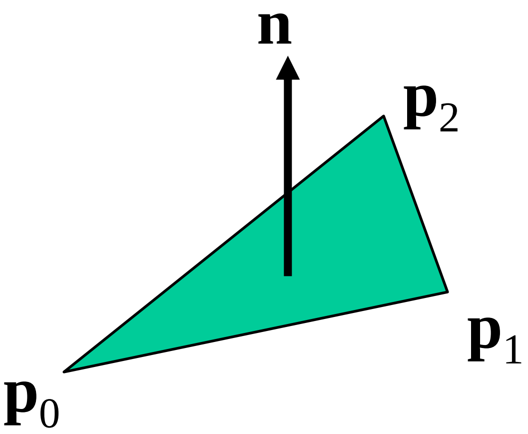
Flat shading

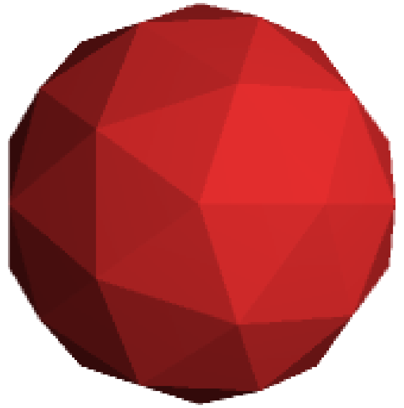
Smooth shading

Review: Flat Shading

- We set <u>a single normal for each triangle</u>

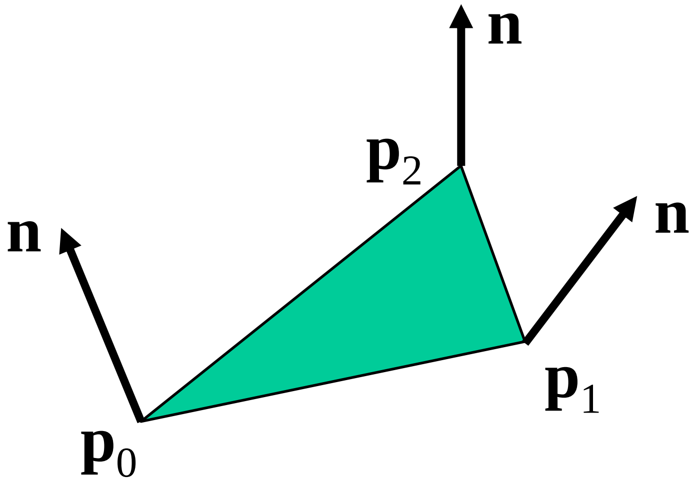
- Because three vertices of a triangle has the same normal, shades computed by the Phong model can be almost same

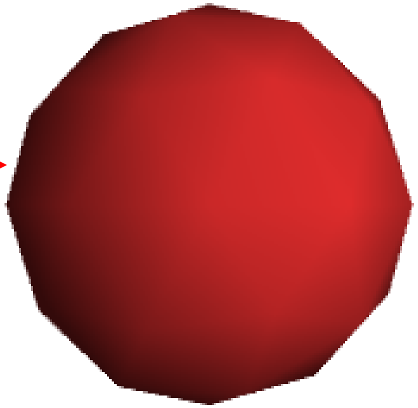




Review: Smooth Shading

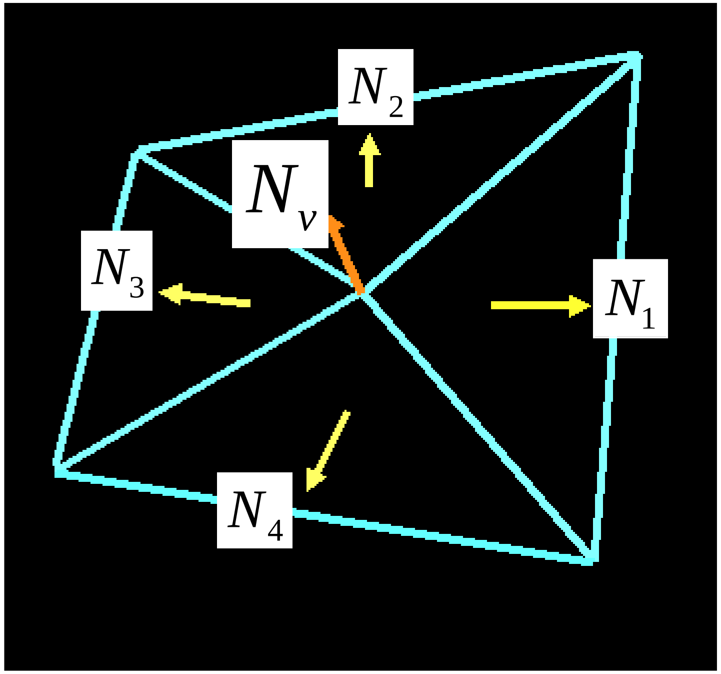
- We set a new normal at each vertex as if the polygon is the approximation of a smooth surface
- For a sphere model, it is easy
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Note *silhouette edge*





Review: Vertex Normal Vector

- Normal vectors at vertices
  - Averaging the normal vectors for each polygon sharing that vertex



$$N_v = \frac{(N_1 + N_2 + N_3 + N_4)}{\left\| N_1 + N_2 + N_3 + N_4 \right\|}$$

Applying Phong Model in two different ways

- Applying Phong model *<u>at each vertex</u>*
  ➔ Gouraud Shading

- Applying Phong model *<u>at each fragment</u>*
  ➔ Phong Shading

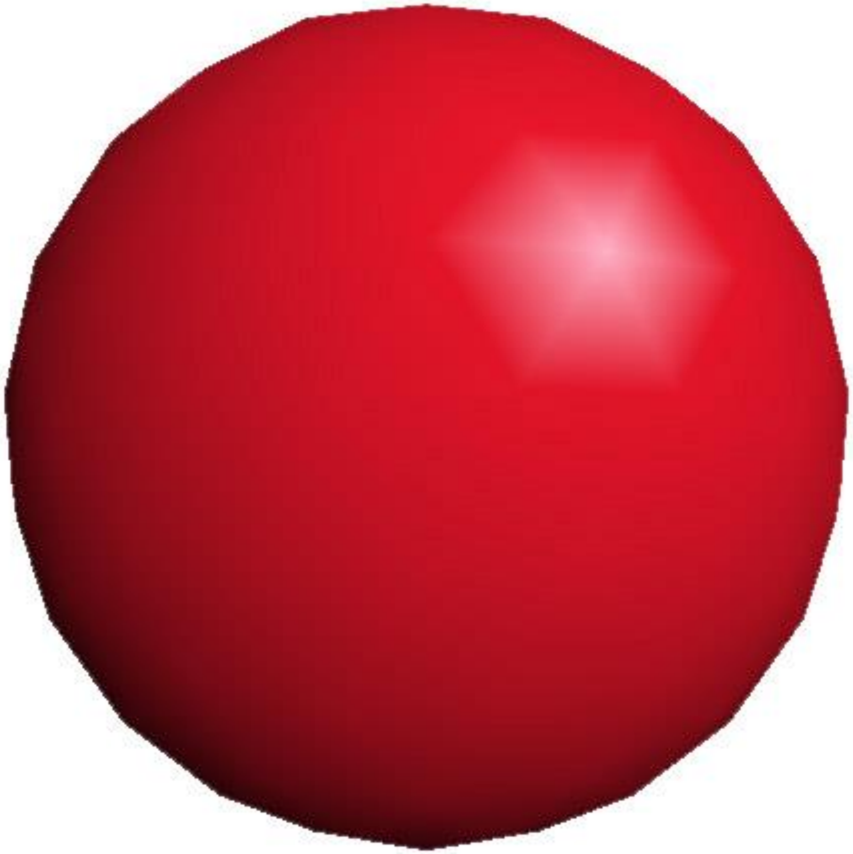
Intensity-Interpolation Surface Rendering

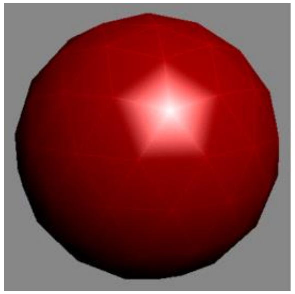
- **Gouraud shading**
  - Rendering a curved surface that is approximated with a polygon mesh
  - Interpolate intensities at polygon vertices

- Procedure
  1. Determine the average unit normal vector at each vertex
  2. Apply an illumination model at each polygon vertex to obtain the light intensity at that position
  3. Linearly interpolate the vertex intensities over the projected area of the polygon

Gouraud Shading Problems

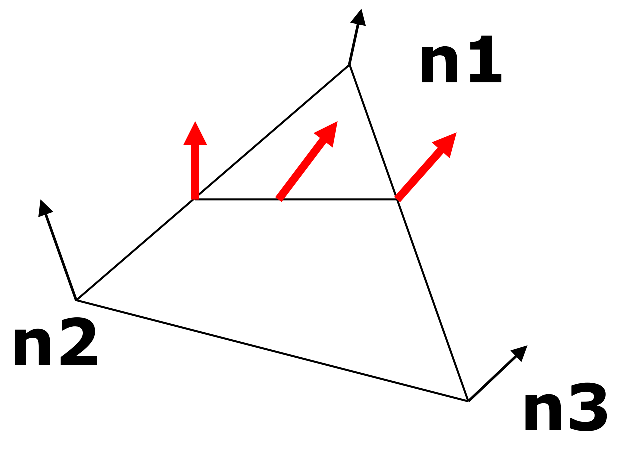
- Lighting in the polygon interior is inaccurate





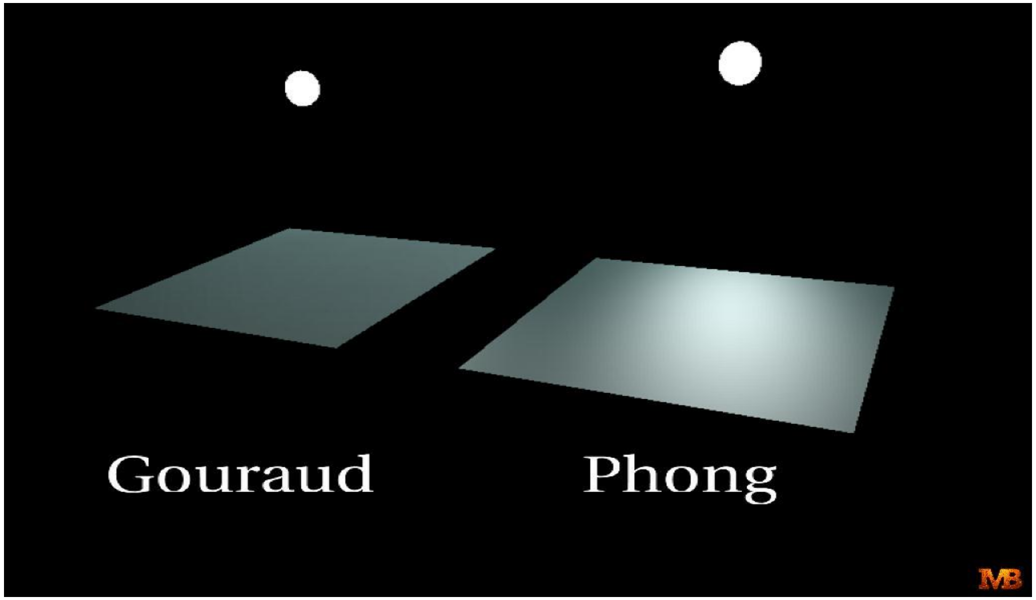
Normal-Vector Interpolation Surface Rendering

- **Phong shading**
  - Interpolate normal vectors at polygon vertices
- Procedure
  1. Determine the average unit normal vector at each vertex
  2. Linearly interpolate the vertex normals over the projected area of the polygon
  3. Apply an illumination model at positions along scan lines to calculate pixel intensities



Gouraud versus Phong Shading

- Gouraud shading is faster than Phong shading
  - OLD OpenGL supports Gouraud shading
- Phong shading is more accurate.
  - Can be implemented using Fragment shader



Gouraud and Phong Shading

- Gouraud Shading
  - Find average normal at each vertex (vertex normals)
  - **Apply Phong model at each vertex**
  - Interpolate vertex shades across each polygon

- Phong shading
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Apply **Phong model at each fragment**

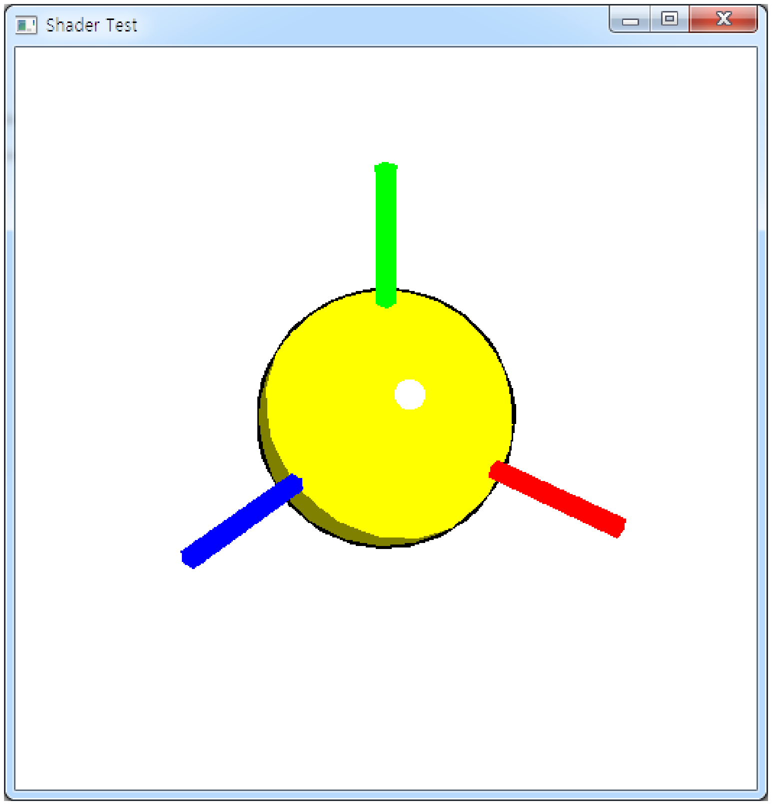
Comparison

- If the polygon mesh approximates surfaces with a high curvatures, Phong shading may look smooth while Gouraud shading may show edges

- Phong shading requires much more work than Gouraud shading
  - Until recently not available in real time systems
  - Now can be done using fragment shaders

- Both need data structures to represent meshes so we can obtain vertex normals

Simple Non-Photorealistic Rendering



Observation:

- Only Two Colors for diffuse
- One Color for Highlights

At Silhouette:

- Black line

Chapter 7.
Discrete Techniques:
Texture Mapping

Texture Mapping

- A way of adding surface details
- Two ways can achieve the goal:
  - Model the surface with more polygons
    - Slows down rendering speed
    - Hard to model fine features
  - Map a texture to the surface
    - This lecture
    - Image complexity does not affect complexity of processing
- Efficiently supported in hardware

Trompe L'Oeil ("Deceive the Eye")

Jesuit Church, Vienna, Austria

- Windows and columns in the dome are painted, not a real 3D object
- Similar idea with texture mapping:
- Rather than modeling the intricate 3D geometry, replace it with an image !

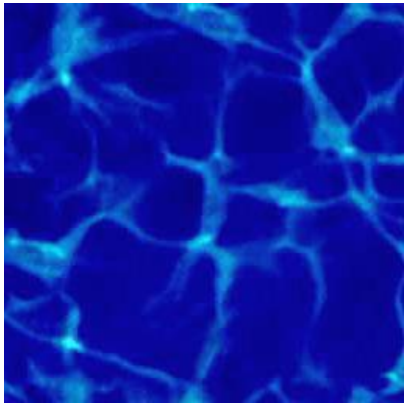
Map textures to surfaces

Texture map

An image

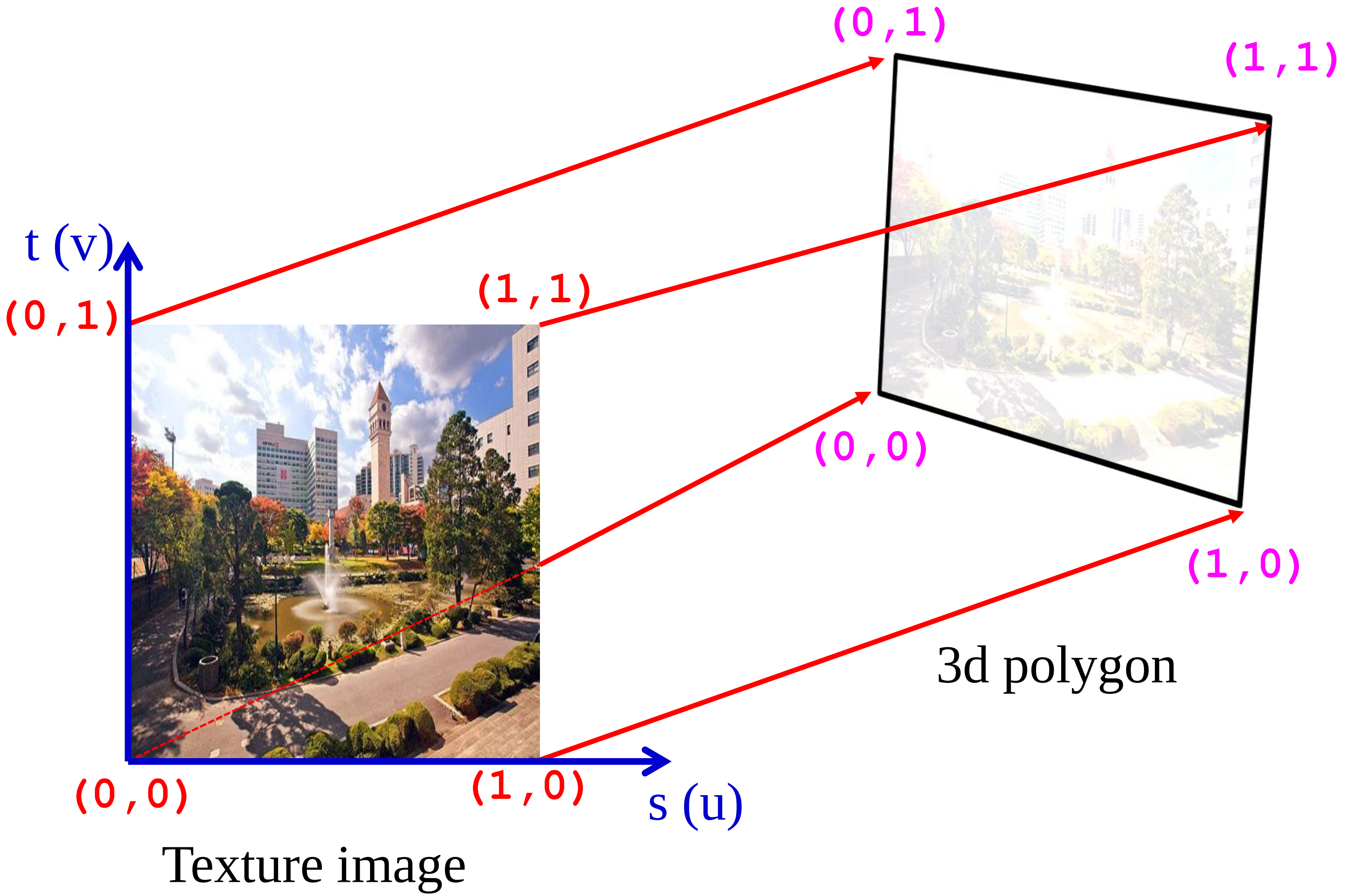
Image mapped to a 3D polygon:
The polygon can have arbitrary size, Shape, and 3D position.

The texture

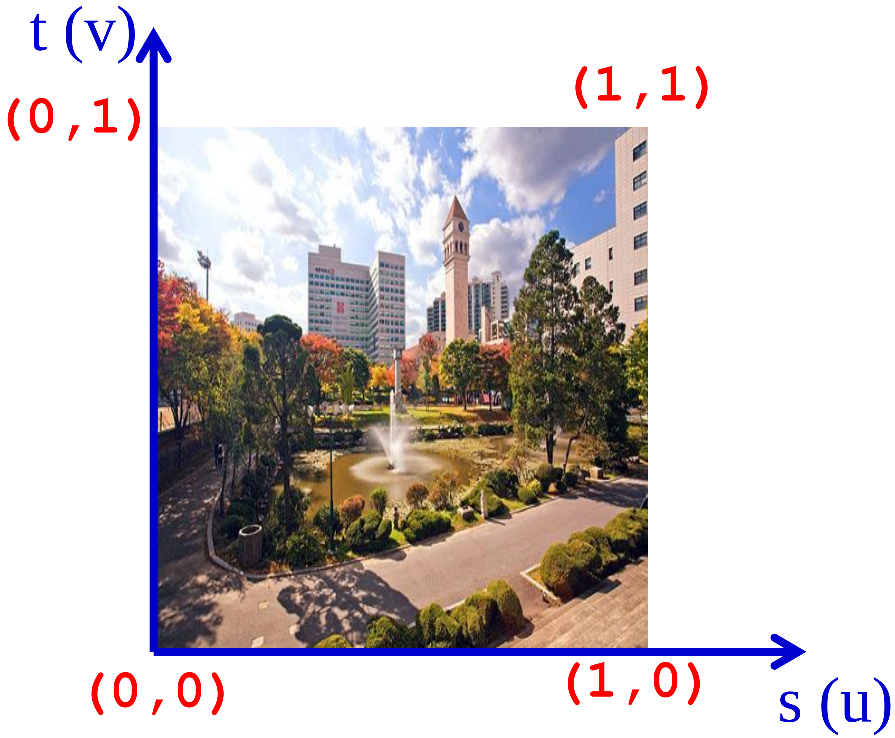


- Texture is a bitmap image
  - Can use an image library to load image into memory
  - Or can create images yourself within the program
- 2D array:
  unsigned char texture[height][width][4]
- Or unrolled into 1D array:
  unsigned char texture[4*height*width]
- Pixels of the texture are called ***texels***
- Texel coordinates (s,t) scaled to [0,1] range
  or also called (u,v) coordinate

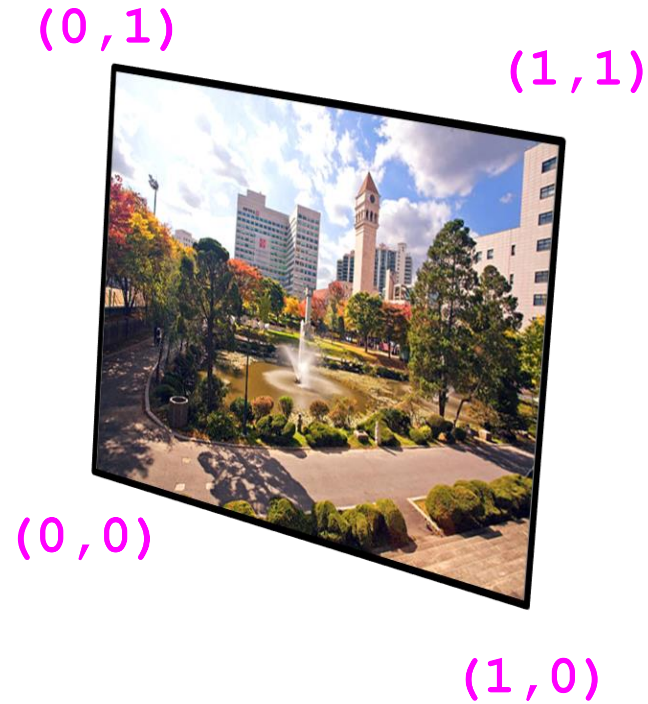
Texture map



Texture map

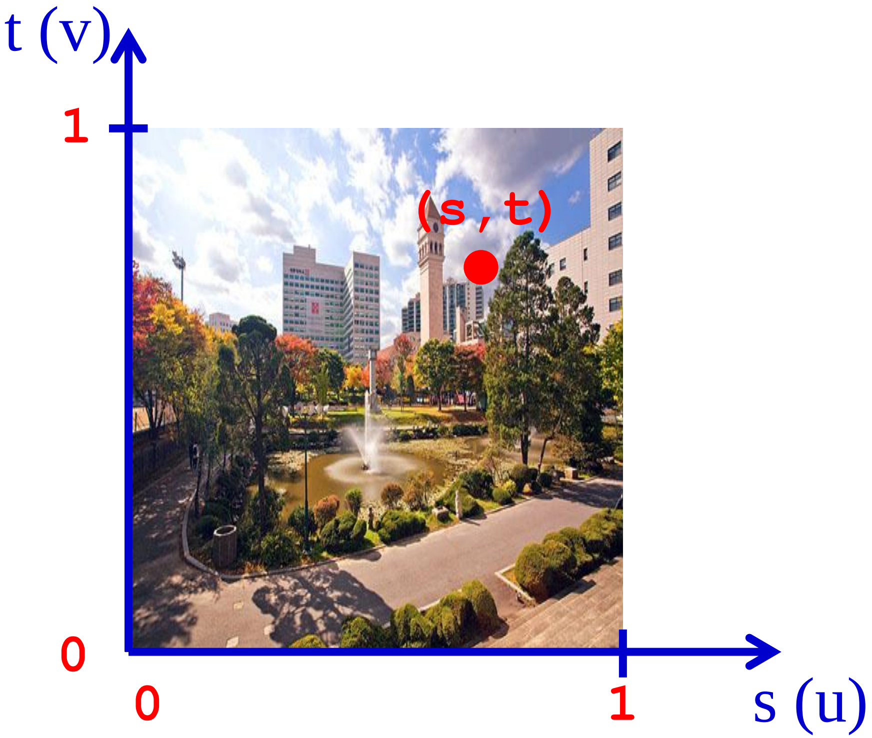


Texture image



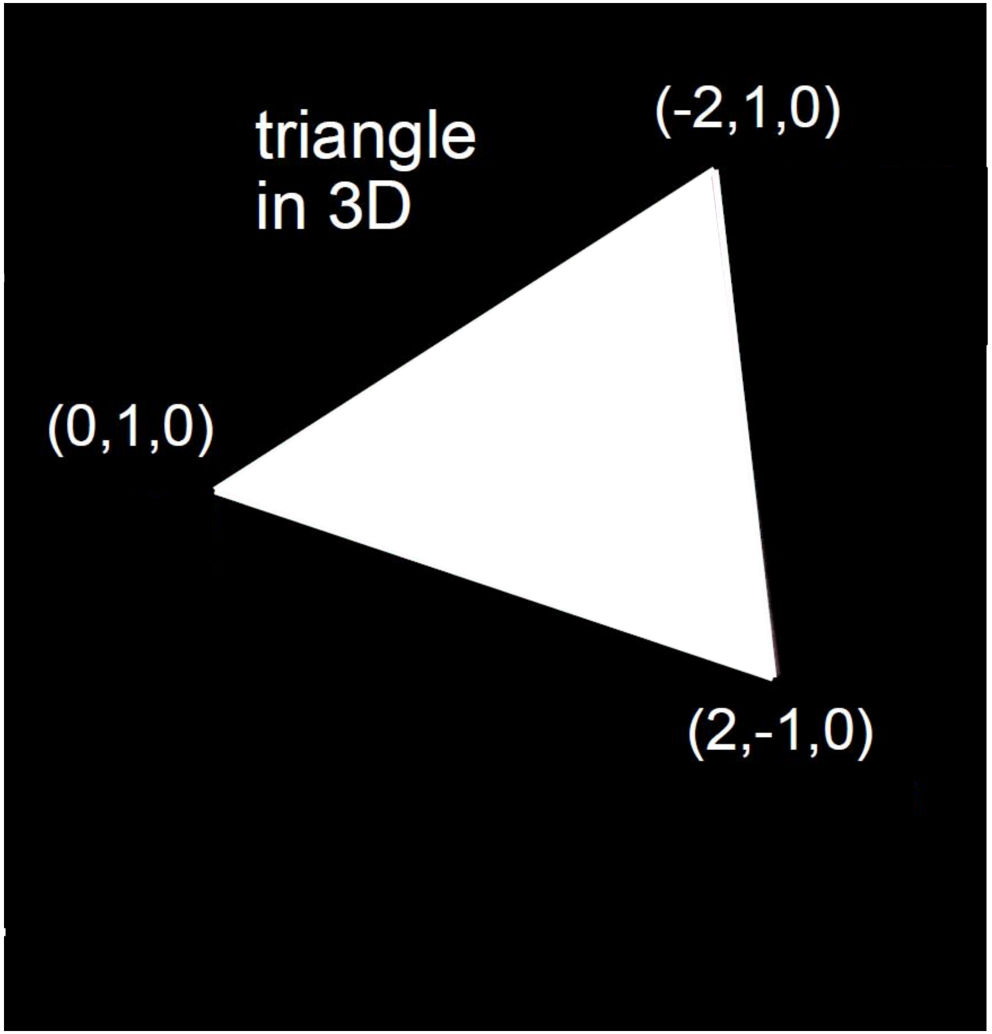
3d polygon

The “st” coordinate system

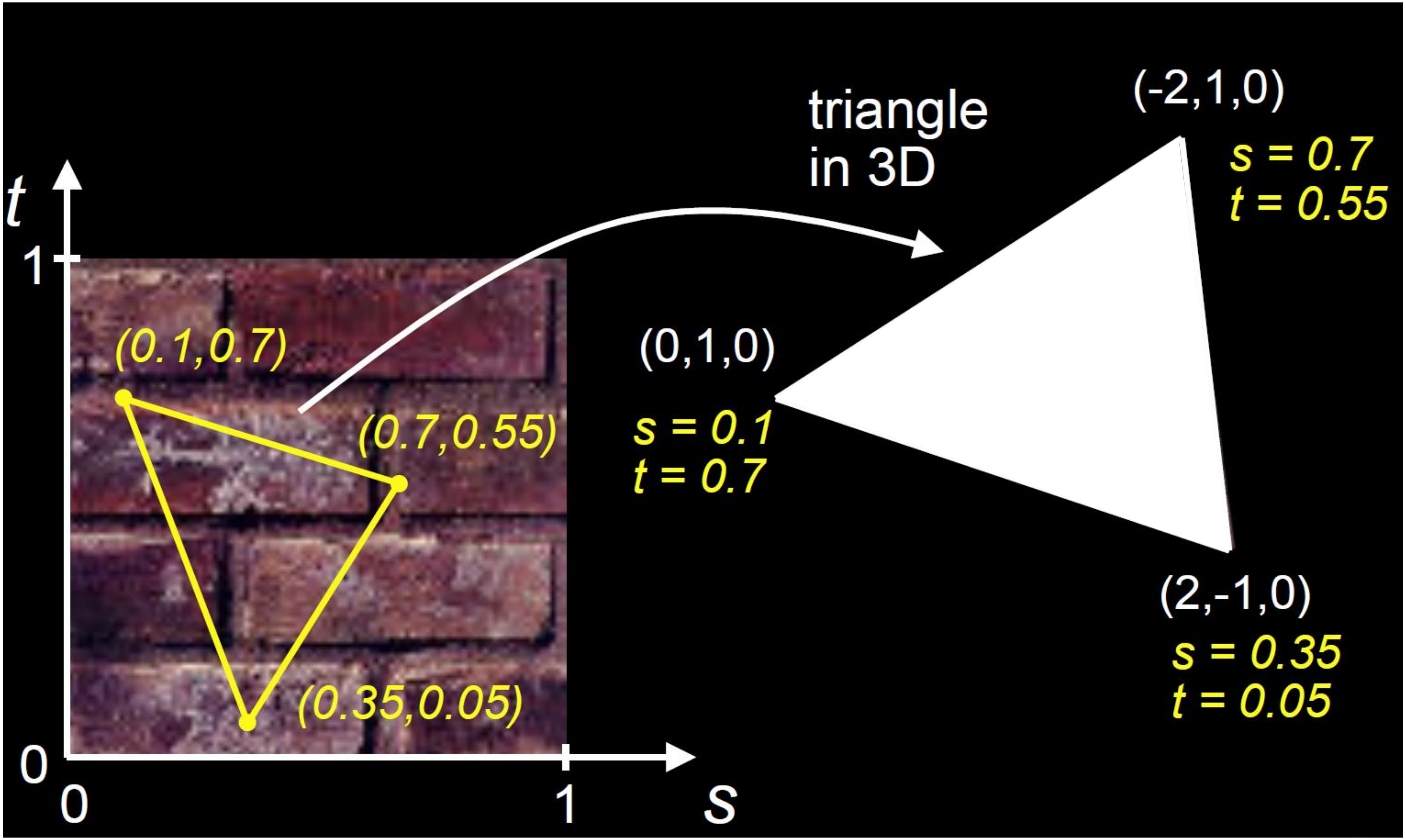


Note: also called “uv” space

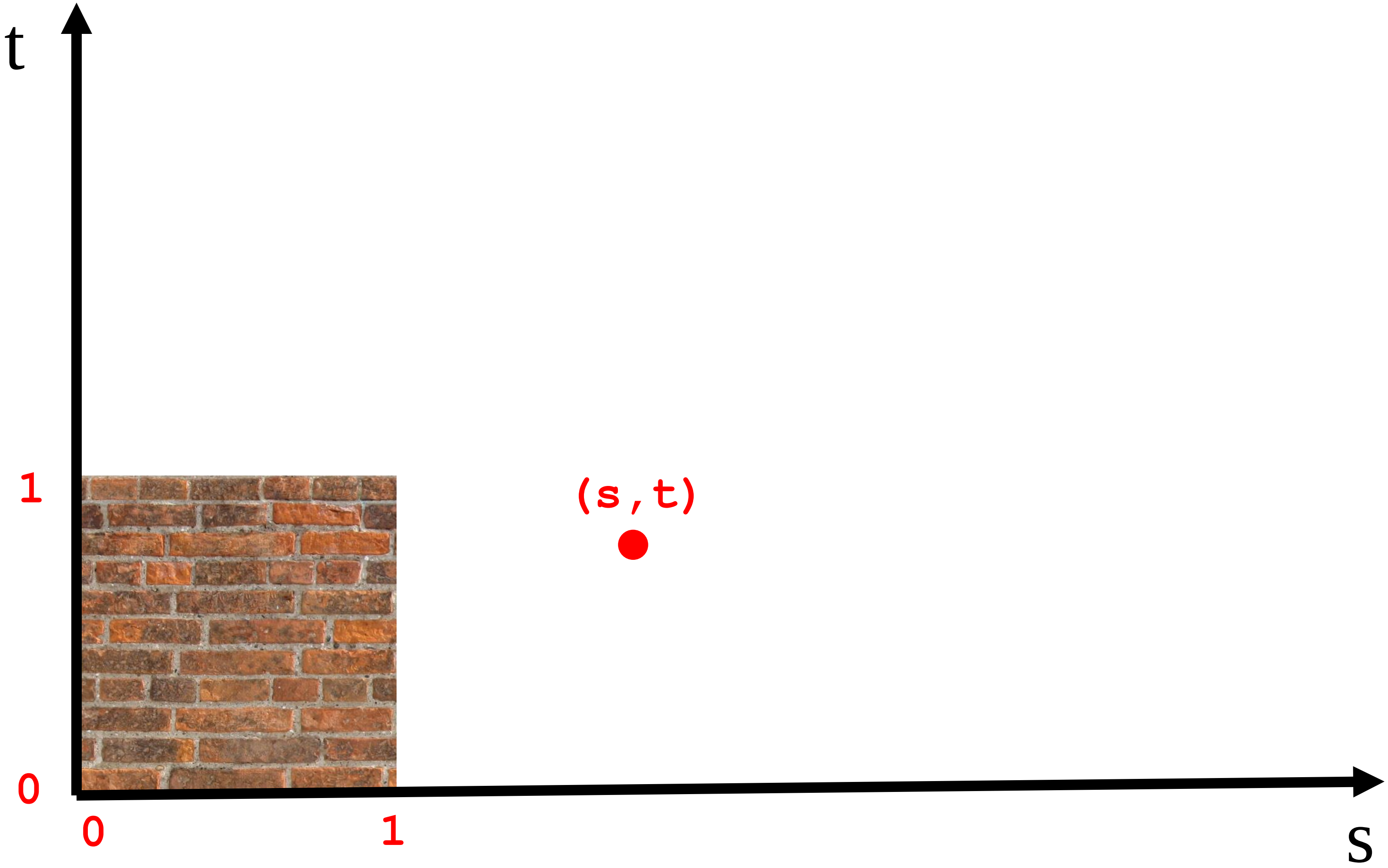
Texture mapping: key slide



Texture mapping: key slide

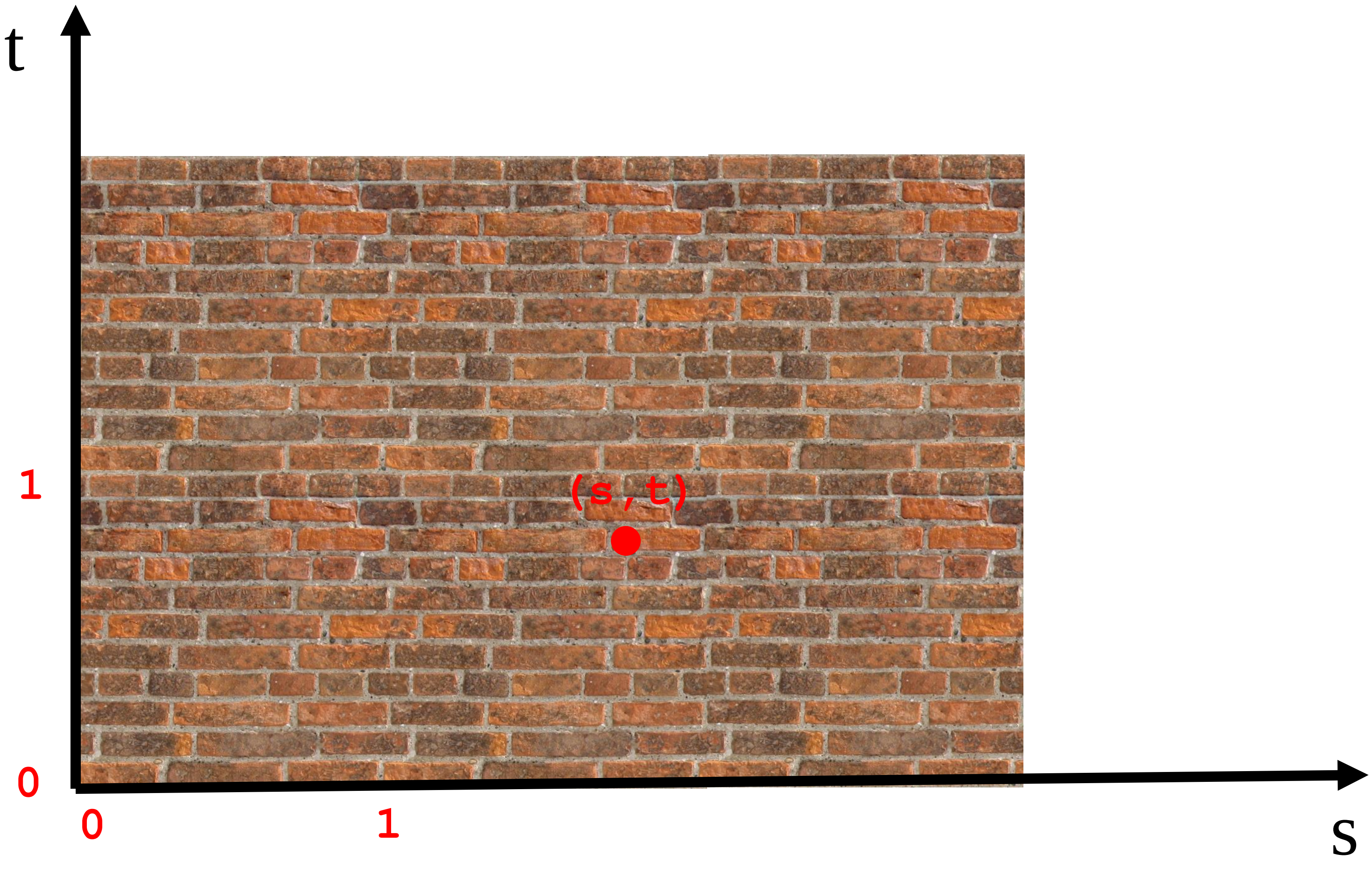


What if texture coordinates are outside of [0,1]



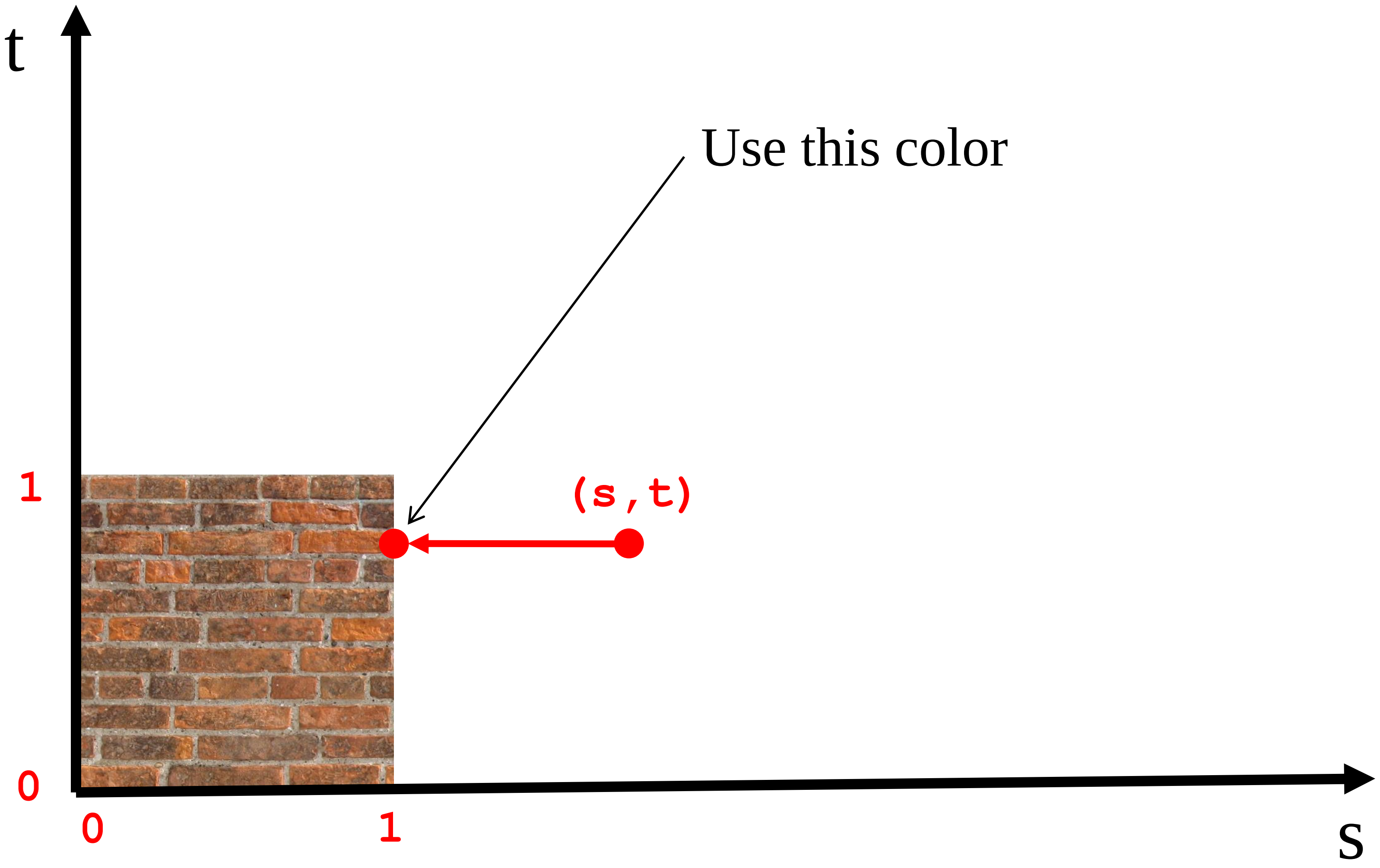
Solution 1: Repeat texture

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



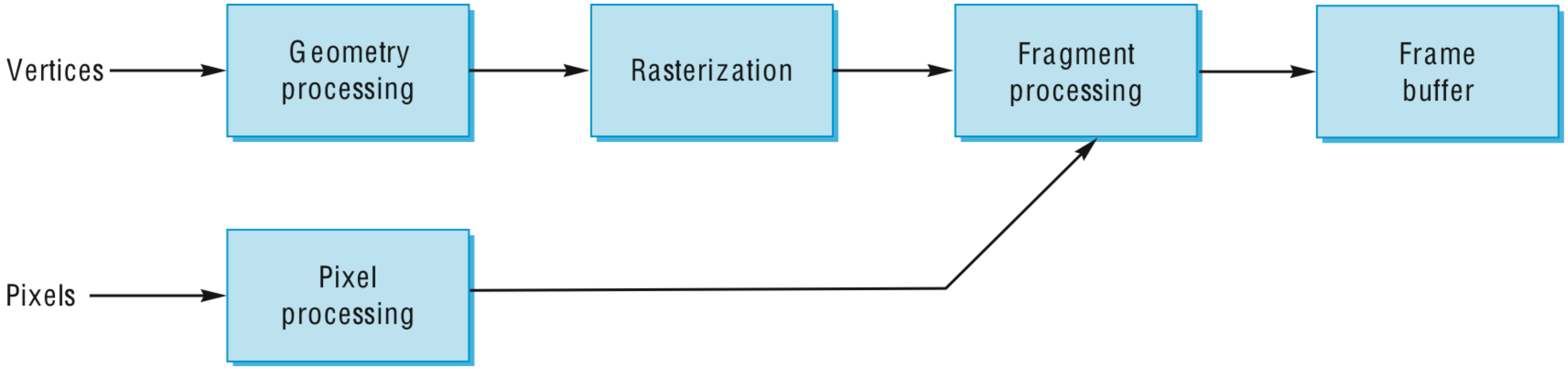
Solution 2: Clamp to [0,1]

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



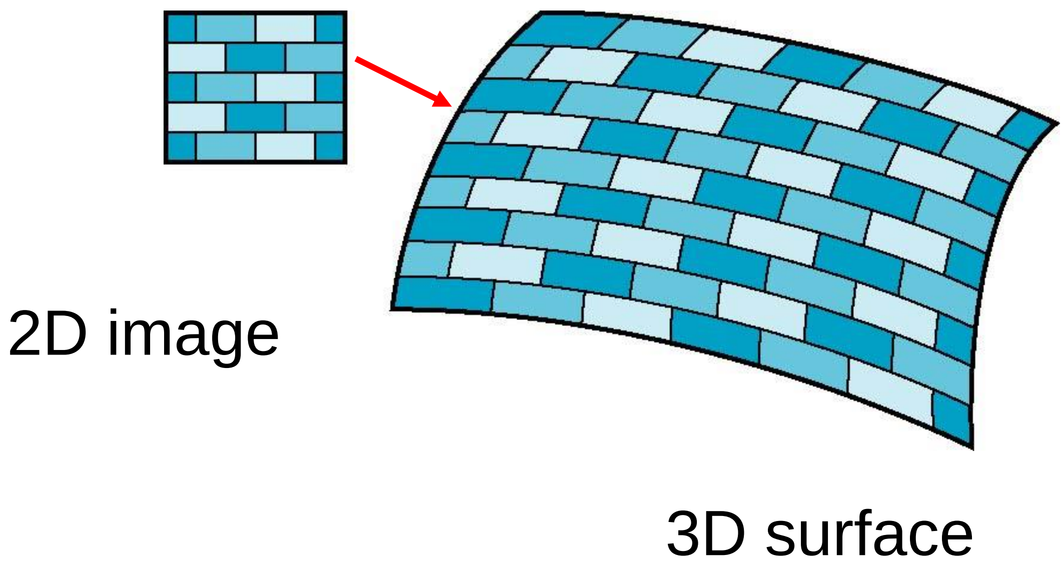
Where does mapping take place?

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



How to compute the map?

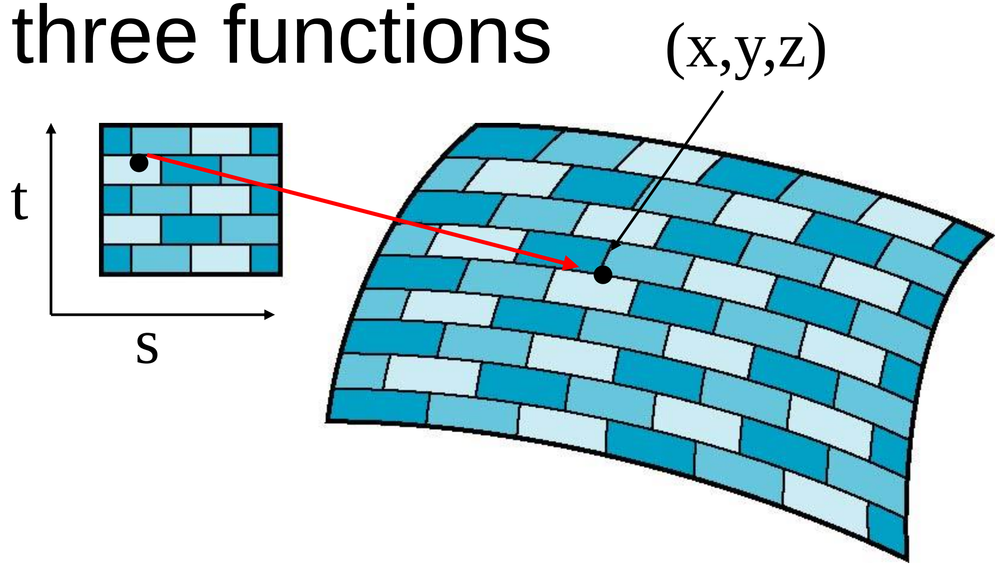
- Is it simple?



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

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



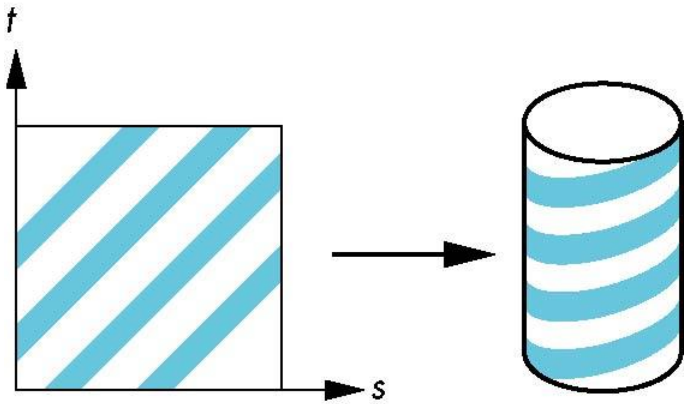
- But we really want to go the other way

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

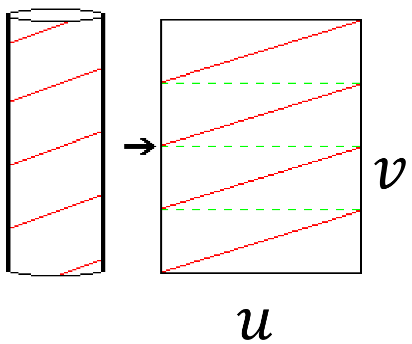
Two-part mapping

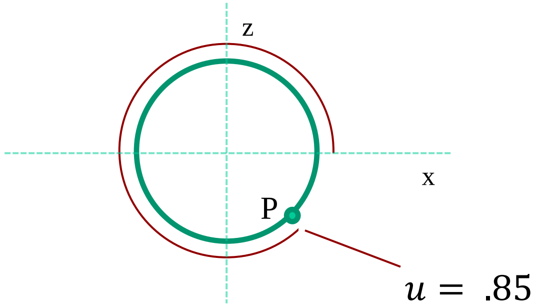
- One solution to the mapping problem is to first map the texture to a simple intermediate surface
- Example: map to cylinder

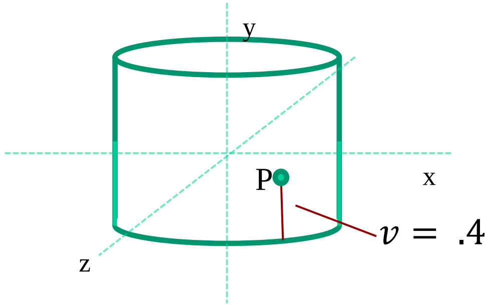


Cylindrical Mapping

- 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

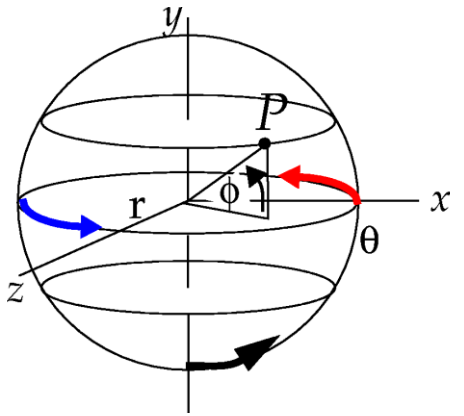






Spherical Map

- 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

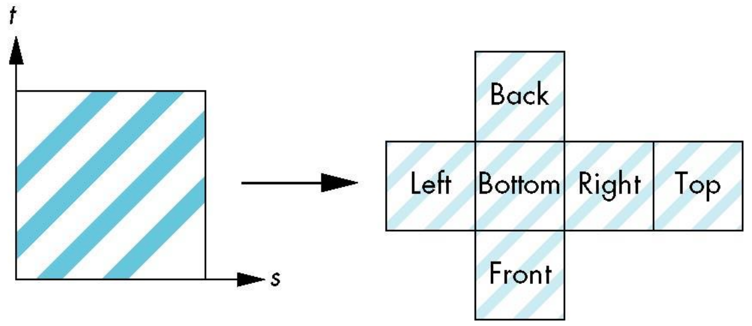


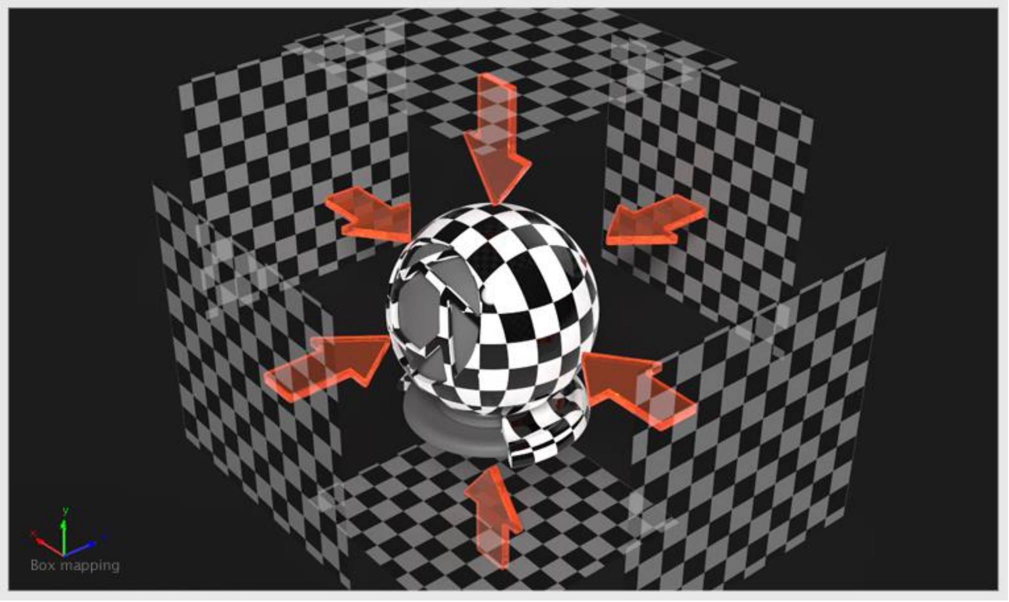
$$\phi = \sin^{-1}\frac{P_y}{r} \qquad -\frac{\pi}{2} \le \phi < \frac{\pi}{2} \qquad r = \text{radius}$$

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

Box Mapping

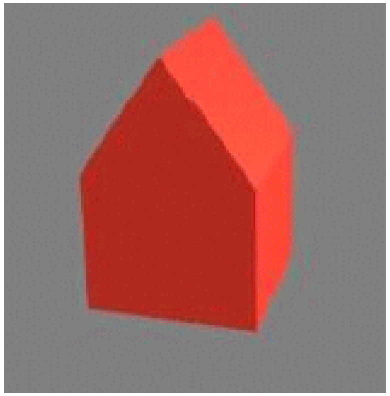
- Easy to use with simple orthographic projection
- Also used in environment maps

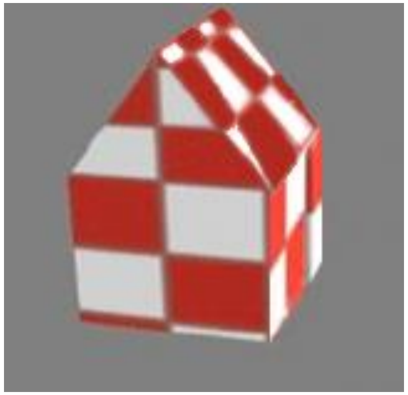




Texture Mapping Complex Geometry

- 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.

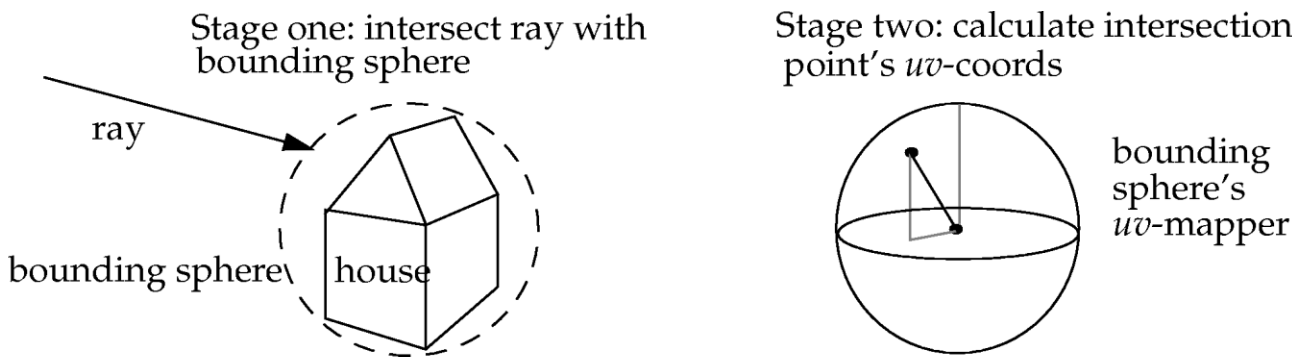




- Solution: Pretend object is a sphere and texture map using the sphere (u, v) map

Texture Mapping Complex Geometry

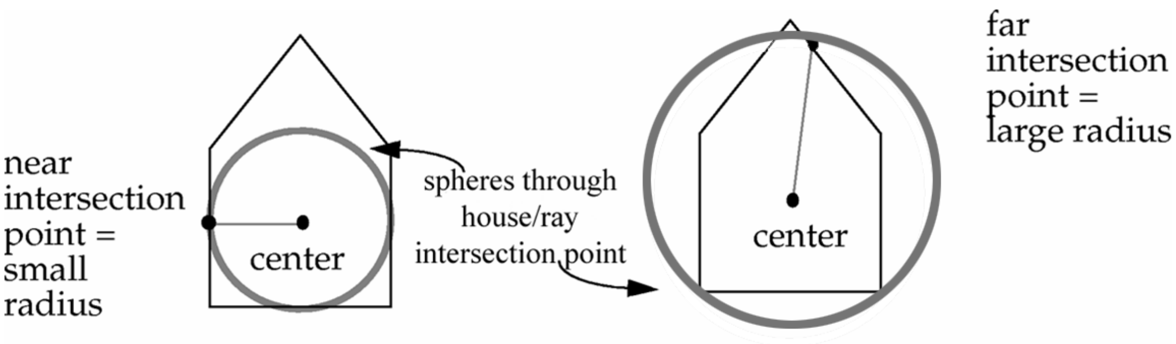
- Intuitive approach: Place a bounding sphere around the complex object
  - Find ray's object space intersection with bounding sphere
  - Convert to (u, v) coordinates



- 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

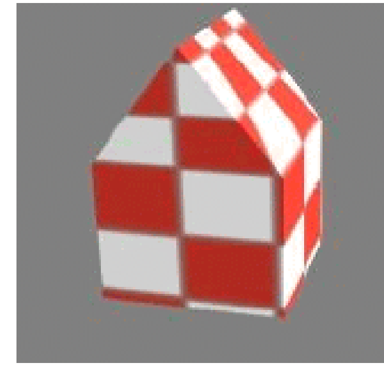
- When we treat the object intersection point as a point on a sphere passing through the point, our “sphere” won’t always have the same radius



- 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:

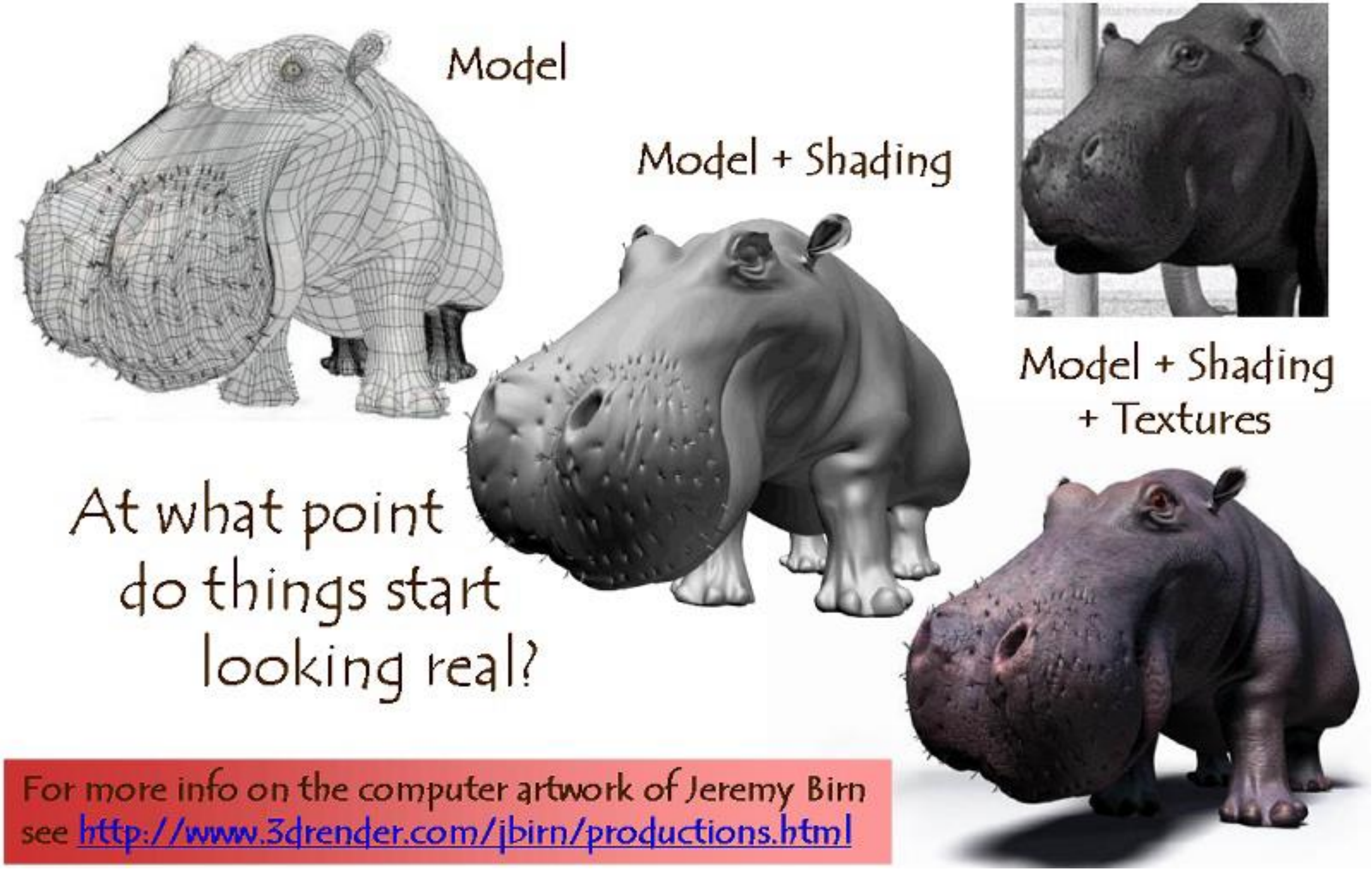


- You can also use cylindrical or planar mappings when texture mapping complex objects
  - Each has drawbacks
    - Spherical: warping at the “poles” of the object (the top of the house)
    - Cylindrical (not shown here): discontinuities at the caps
    - Planar: one axis is ignored

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);
```