Chapter 7

Texture Mapping in OpenGL

Interactive Computer Graphics

Reference:

An interactive introduction to OpenGL programming (SIGGRAPH 2007 Course) by Ed Angel, Dave Shreiner, and Vicki Shreiner

CSCI 420: Computer Graphics FS 2018 by Prof. Dr. Hao Li

Chakrit Watcharopas

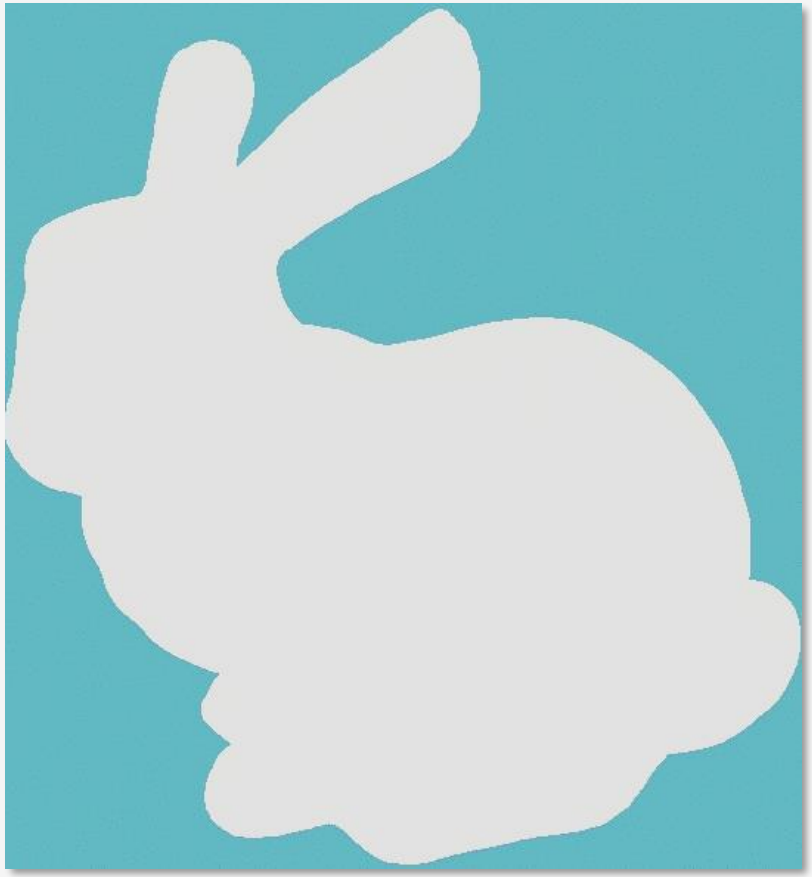
Outline

- Texture Mapping in OpenGL
- Filtering and Mipmaps
- Example of Code
- Demo

Texture Mapping in OpenGL

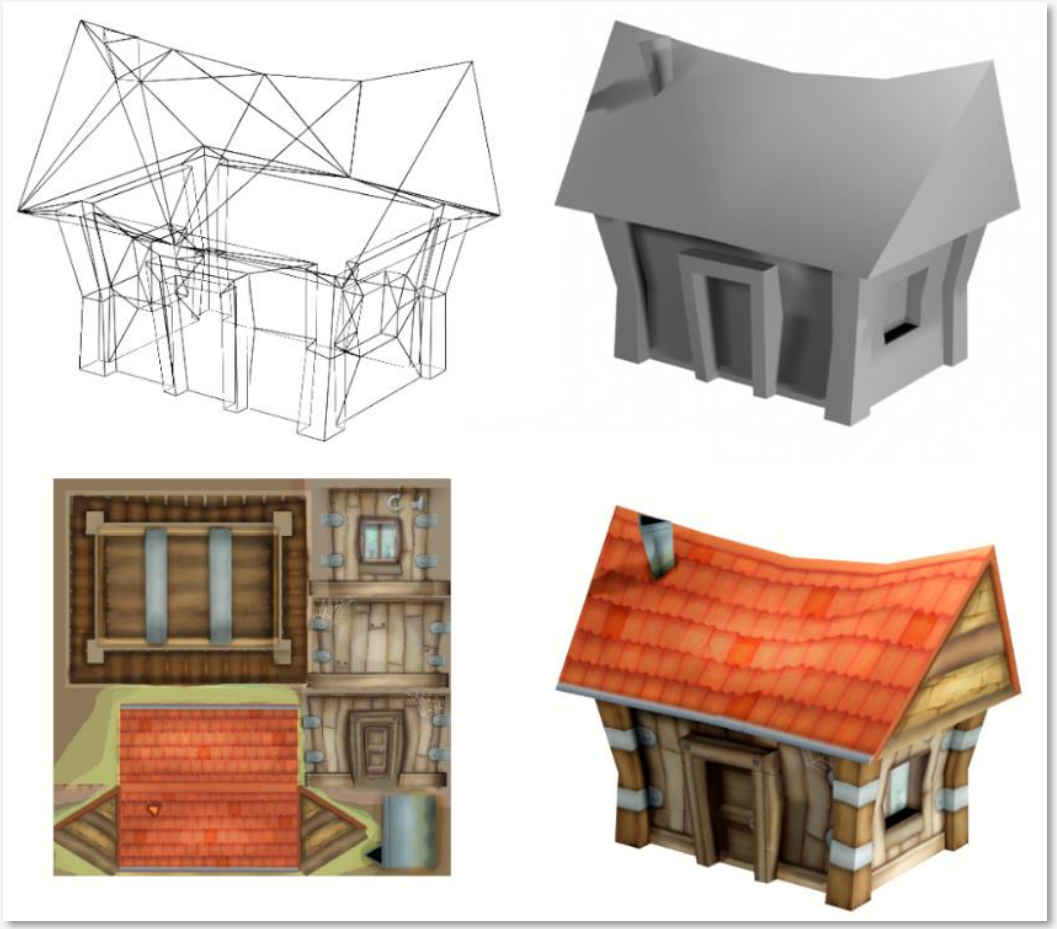
How can we add detail to a bunny

- The left bunny is rendered in solid gray, while the right bunny is mapped with a texture



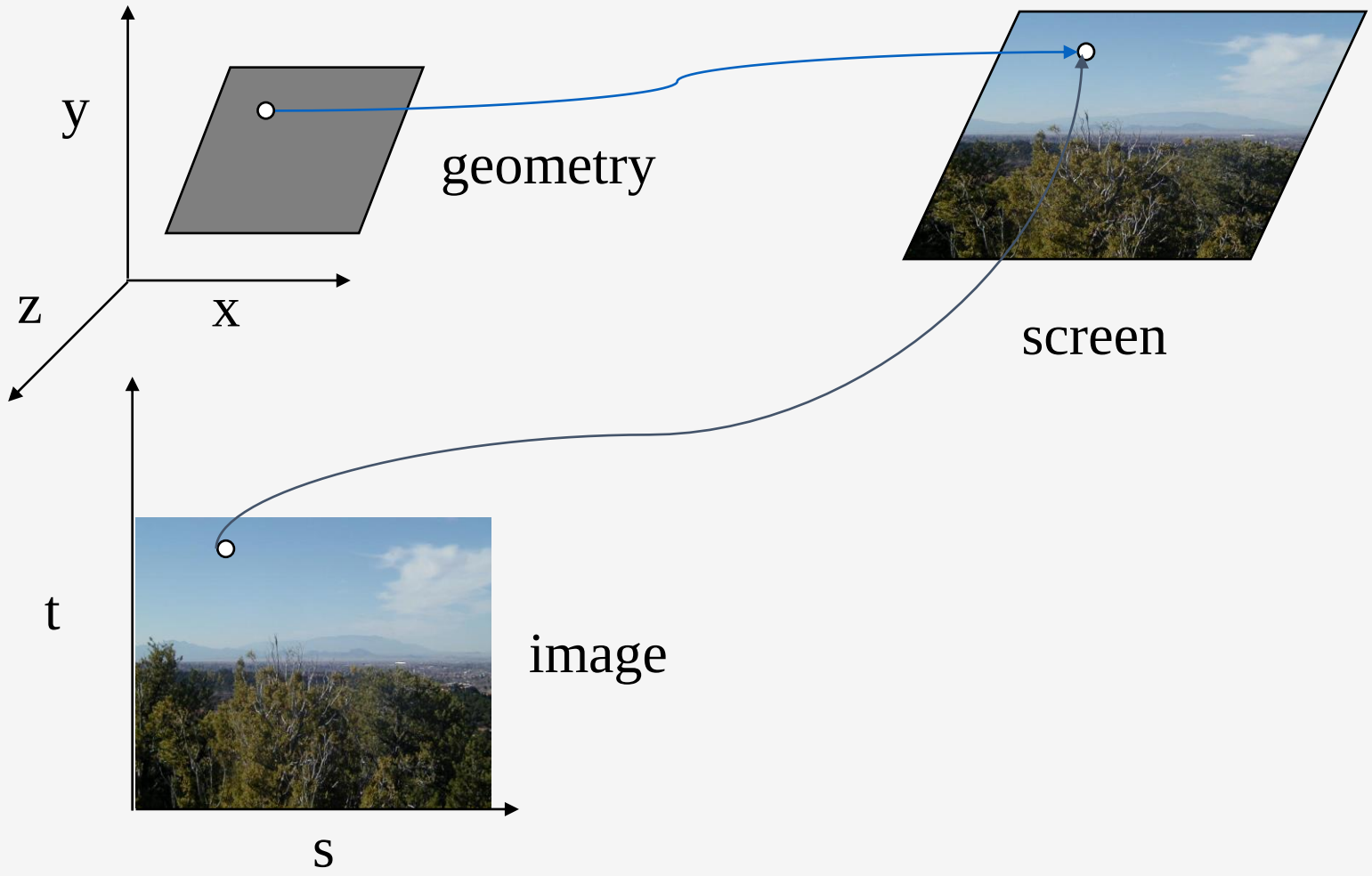
Texture Mapping

- A way of adding surface details
- Two ways can achieve the goal:
  - Model a surface with more polygons
    - slows down rendering speed
    - hard to model fine features
  - Map a texture to the surface
    - this week lesson
    - image complexity does not affect complexity of processing
- Efficiently supported in hardware



Source: https://en.wikipedia.org/wiki/Texture_mapping

Map Textures to Surfaces

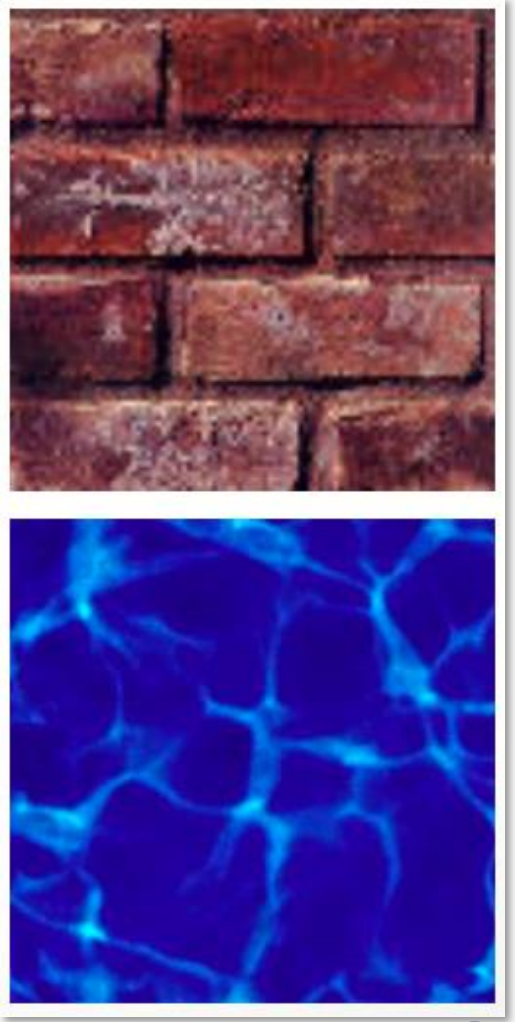


The Texture

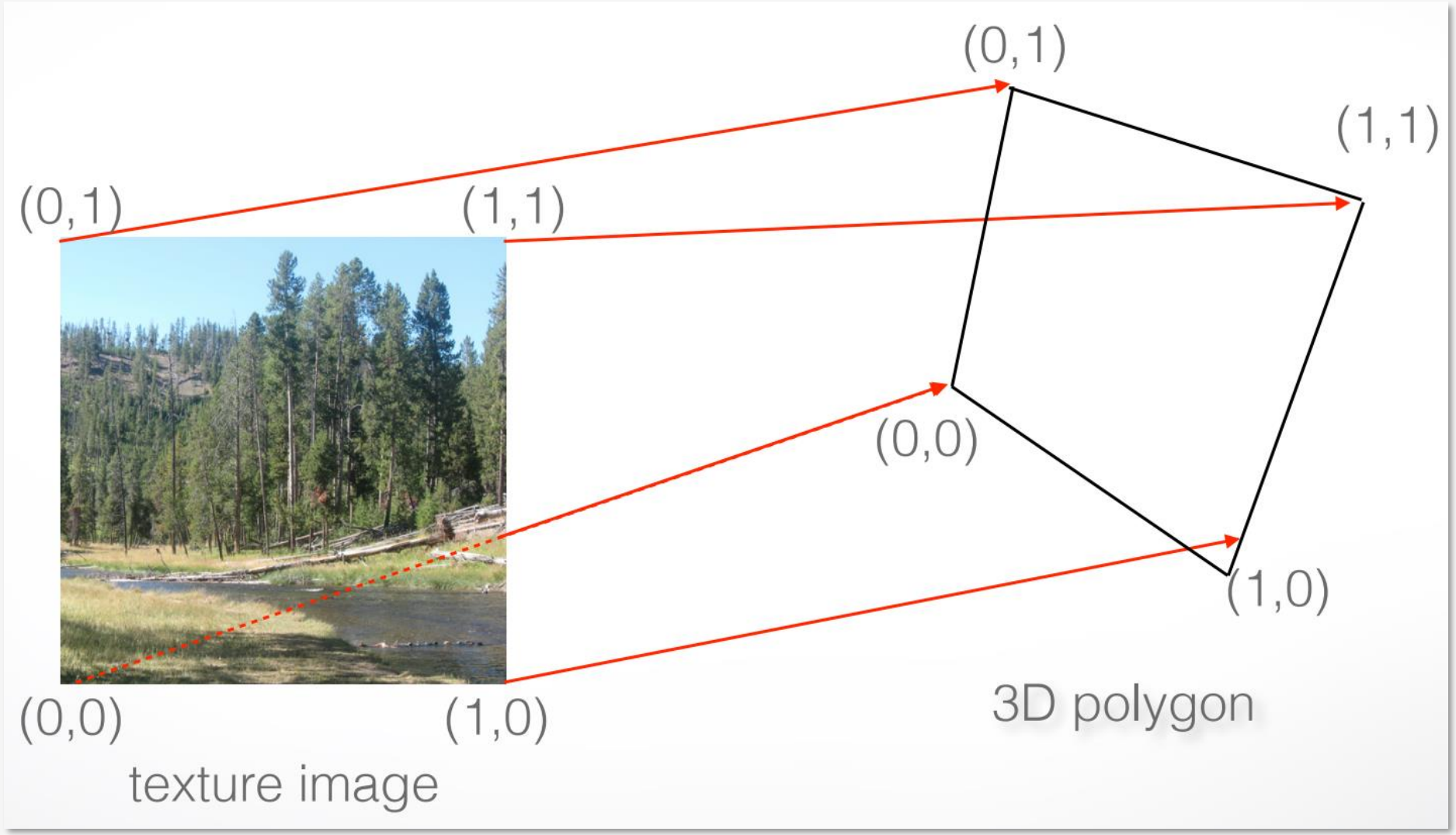
- Texture is a bitmap image
  - can use an image library to load image into memory
  - or can create images yourself within the program
- 2D array:

  np.zeros((height, width, 3), dtype=np.uint8)
- Or unrolled into 1D array:

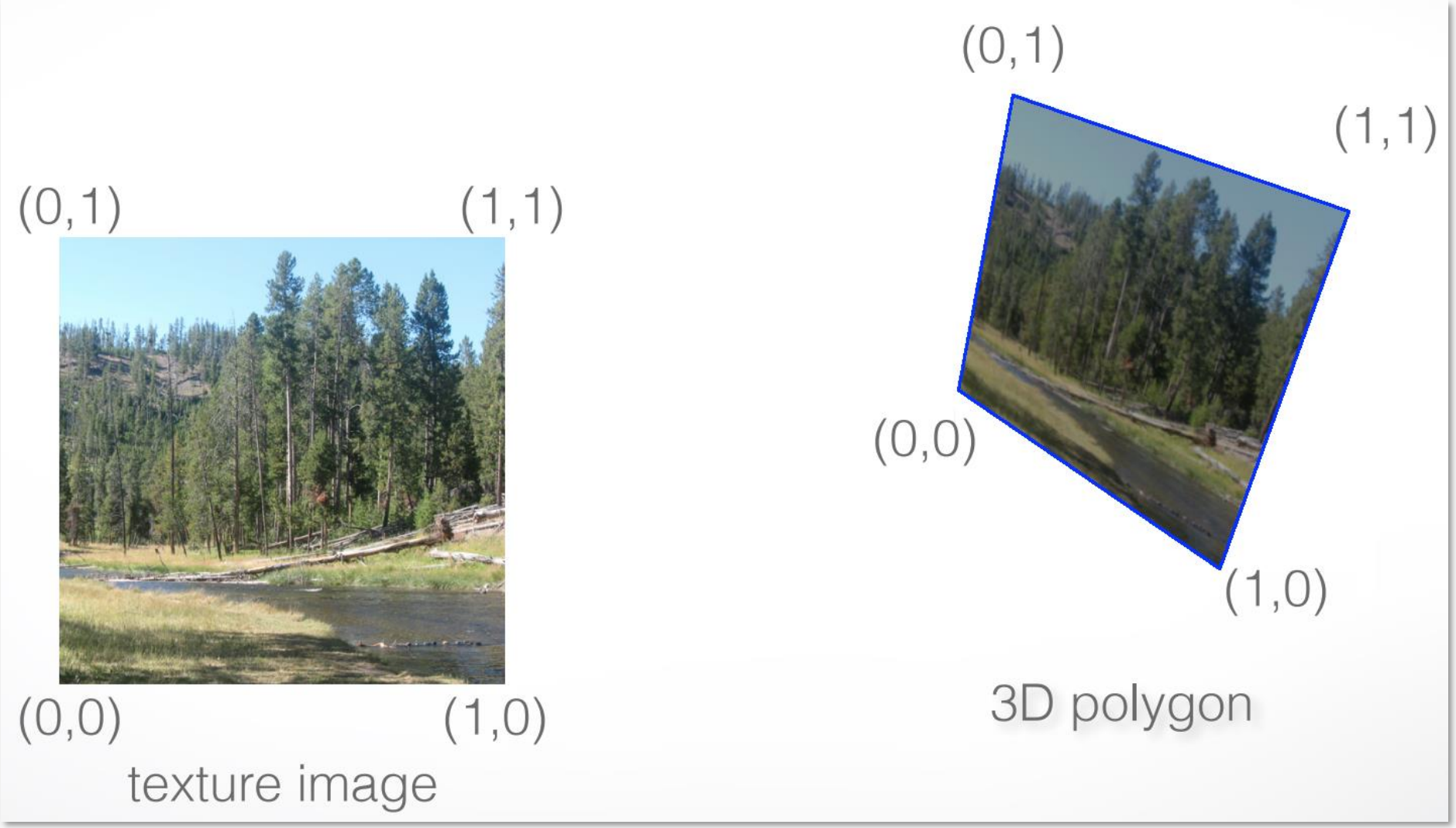
  np.zeros(3*width*height, dtype=np.uint8)
- Pixels of the texture are called *texels*
- Texel coordinates (s, t) scaled to [0,1] range



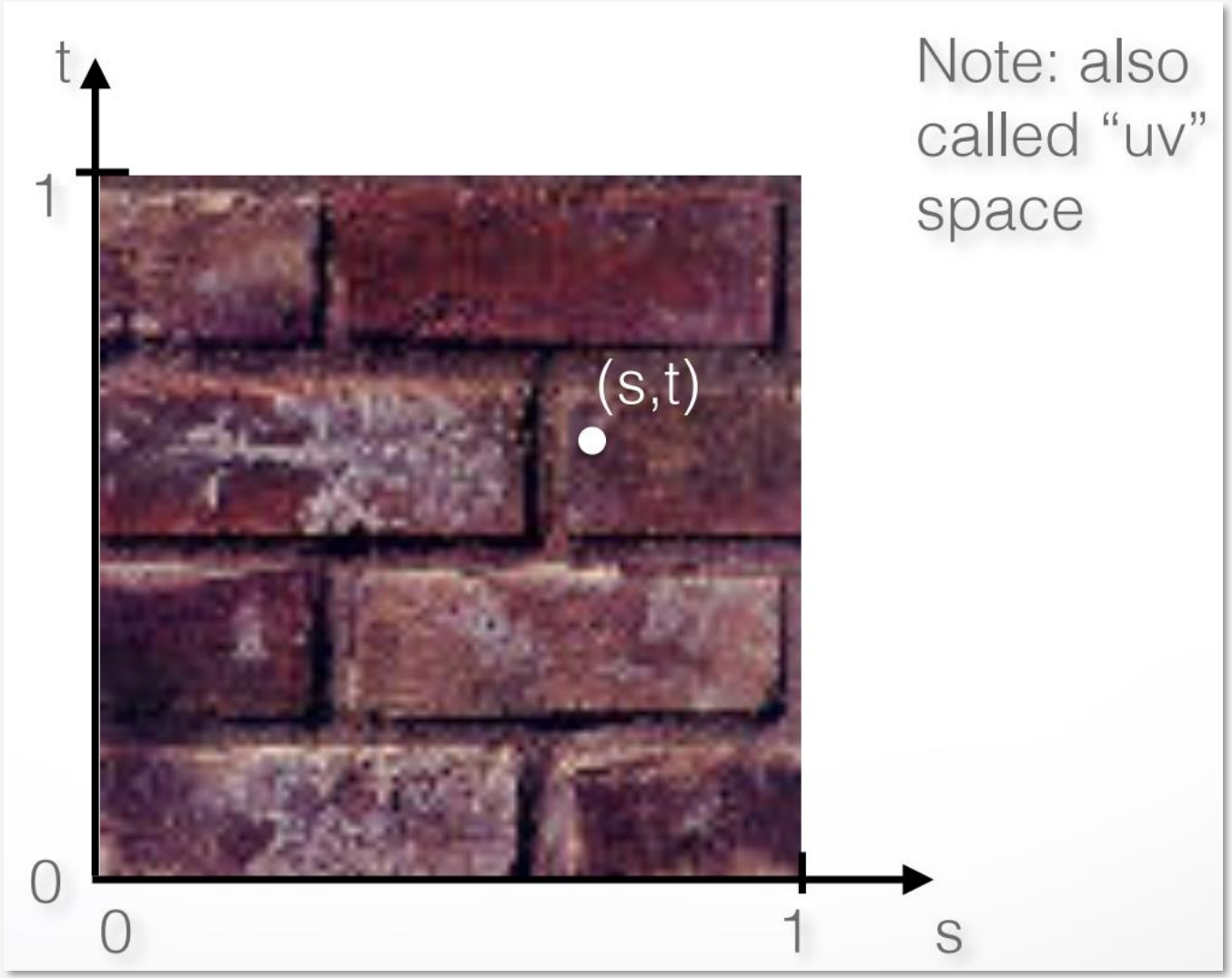
Texture Map



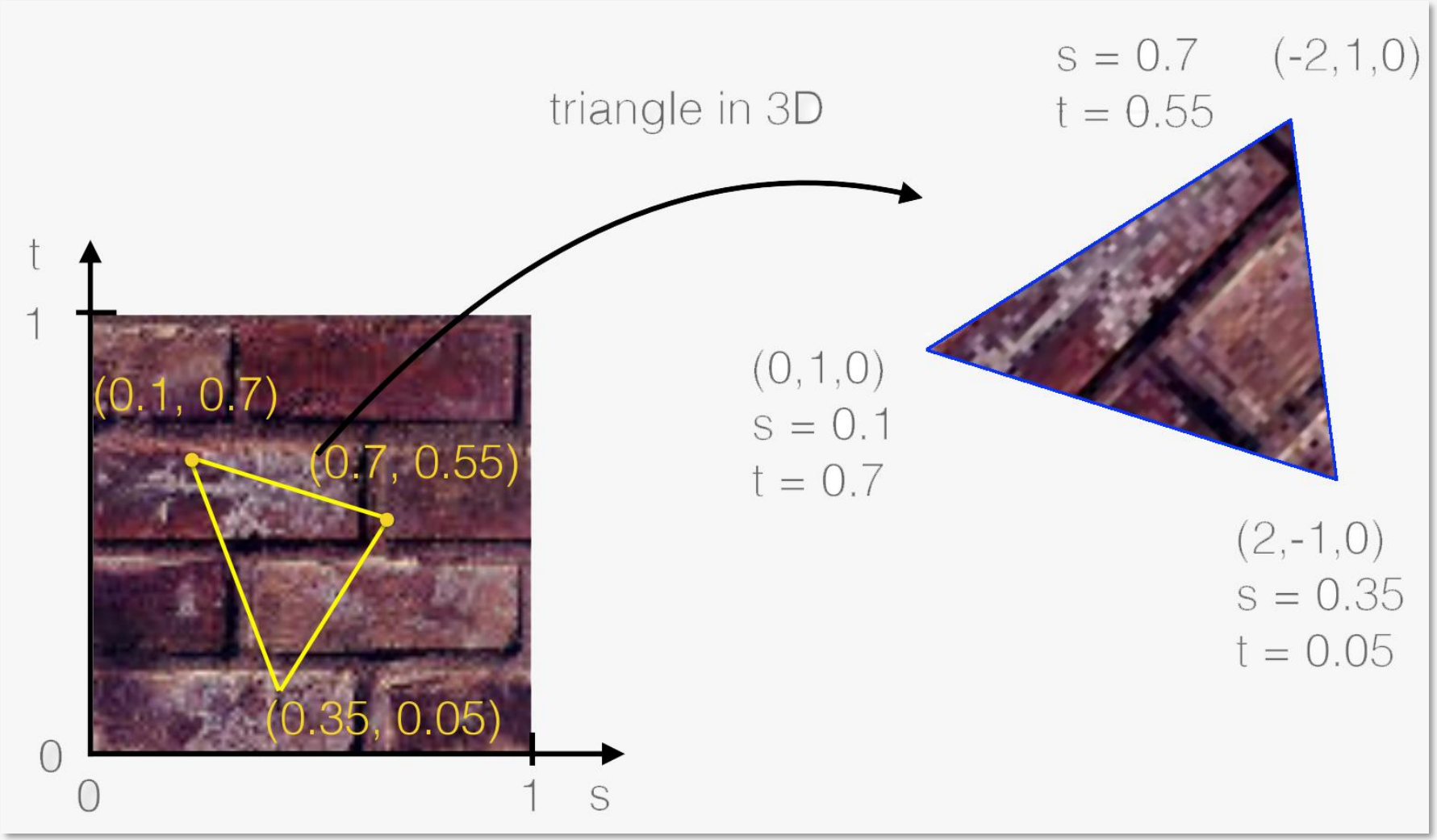
Texture Map



The “st” Coordinate System

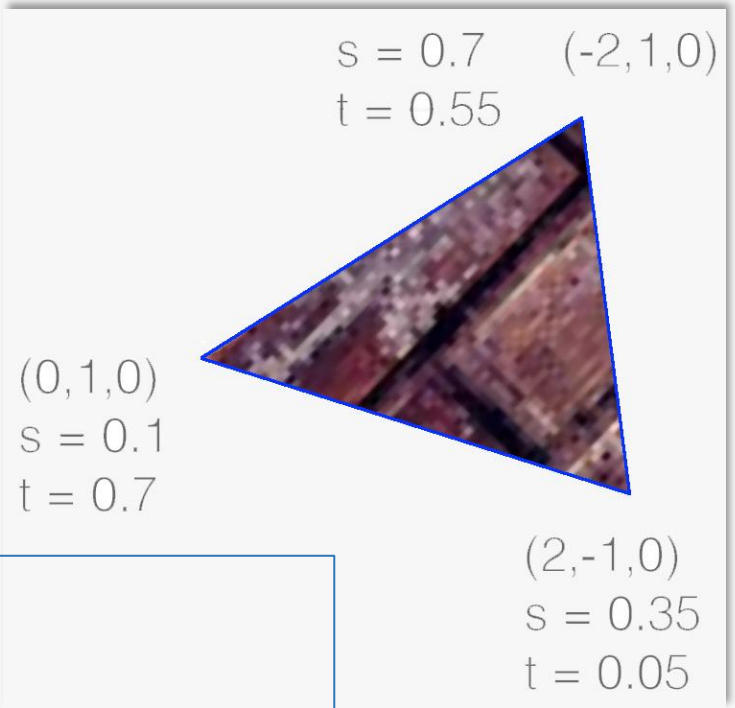


Texture Mapping



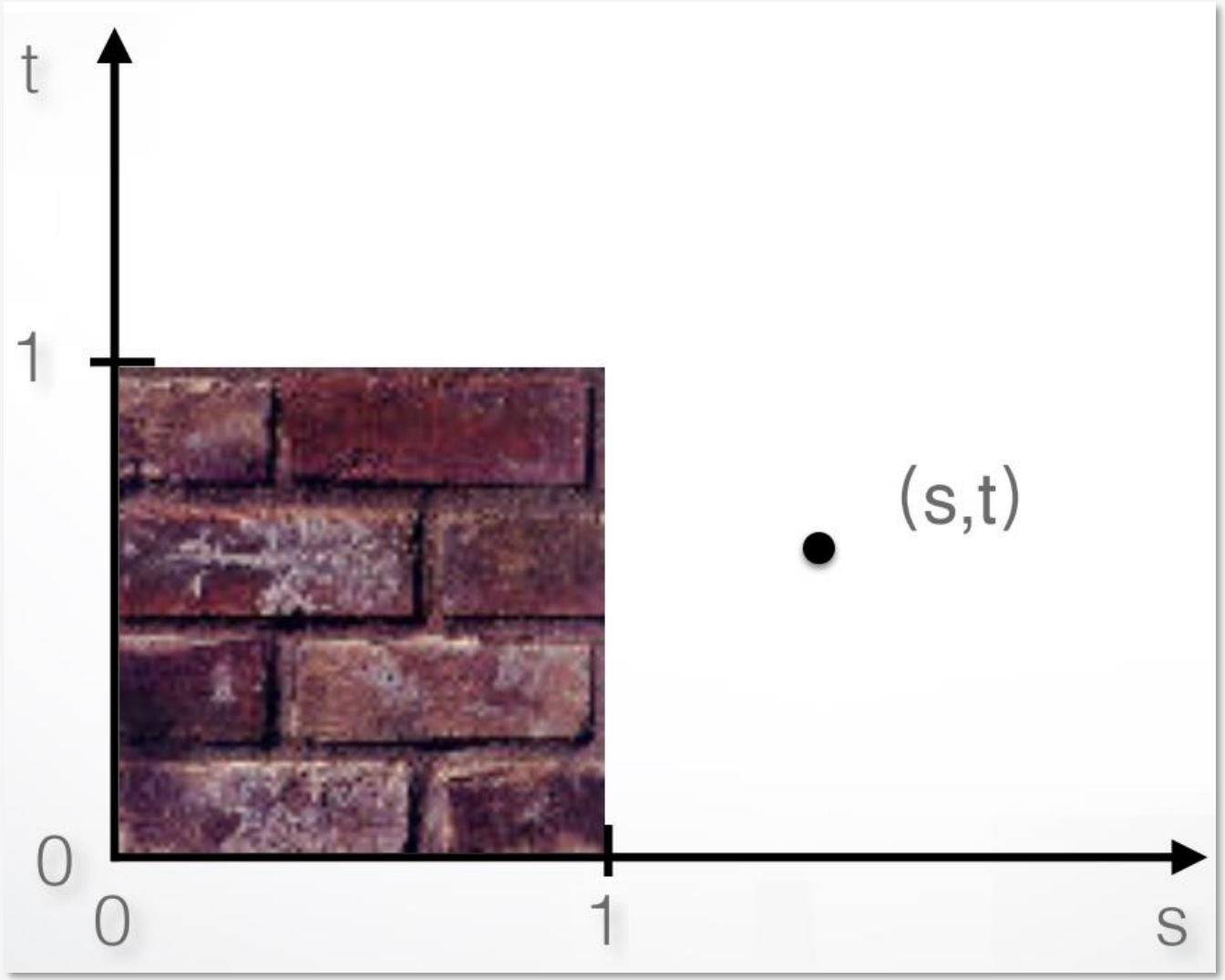
Specifying Texture Coordinates in OpenGL

- Use `glTexCoord2f(s, t)`
- State machine: texture coordinates remain valid until you change them
- Example (from previous slide)



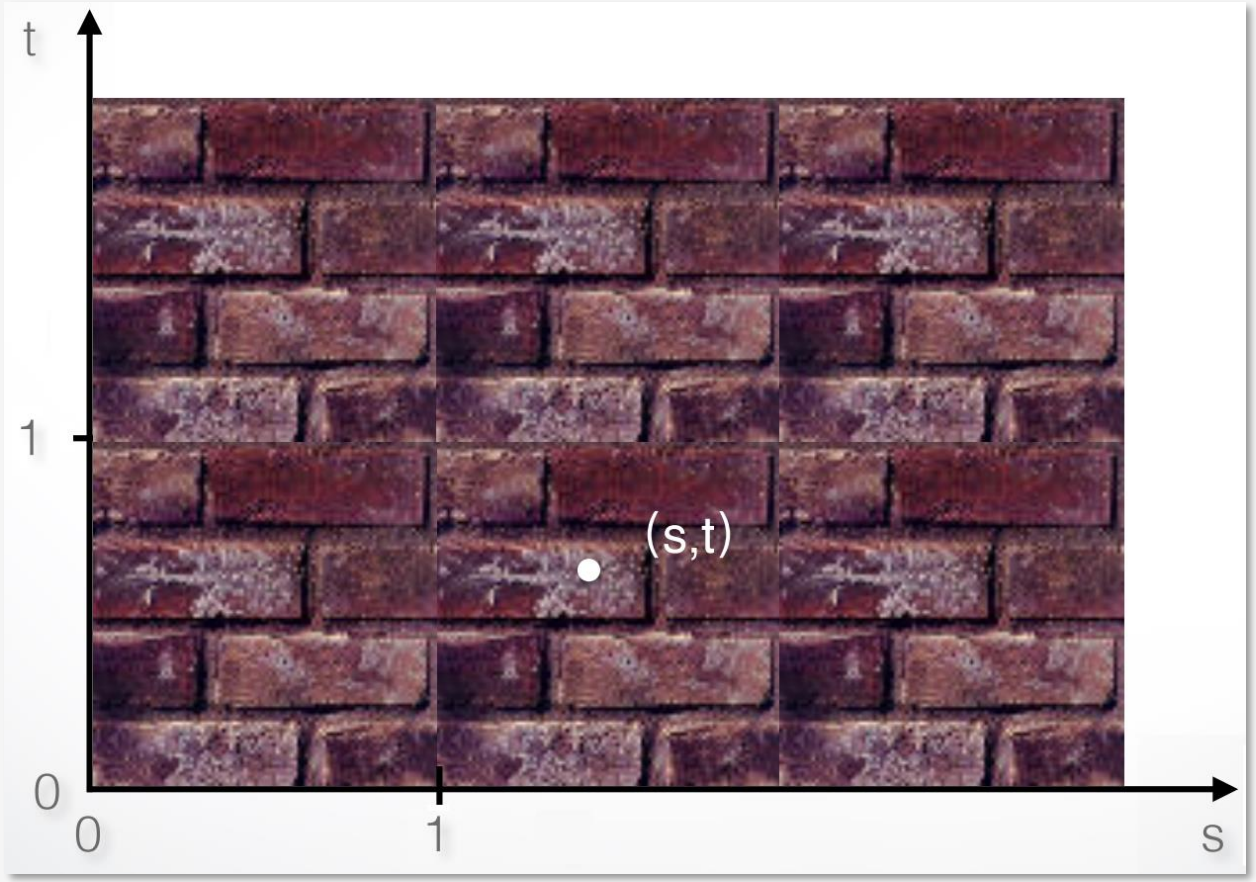

```
glEnable(GL_TEXTURE_2D)
glBegin(GL_TRIANGLES)
glTexCoord2f(0.35, 0.05); glVertex3f(2.0, -1.0, 0.0)
glTexCoord2f(0.7, 0.55); glVertex3f(-2.0, 1.0, 0.0)
glTexCoord2f(0.1, 0.7); glVertex3f(0.0, 1.0, 0.0)
glEnd()
glDisable(GL_TEXTURE_2D)
```

What if Texture Coordinates are Outside of [0, 1]



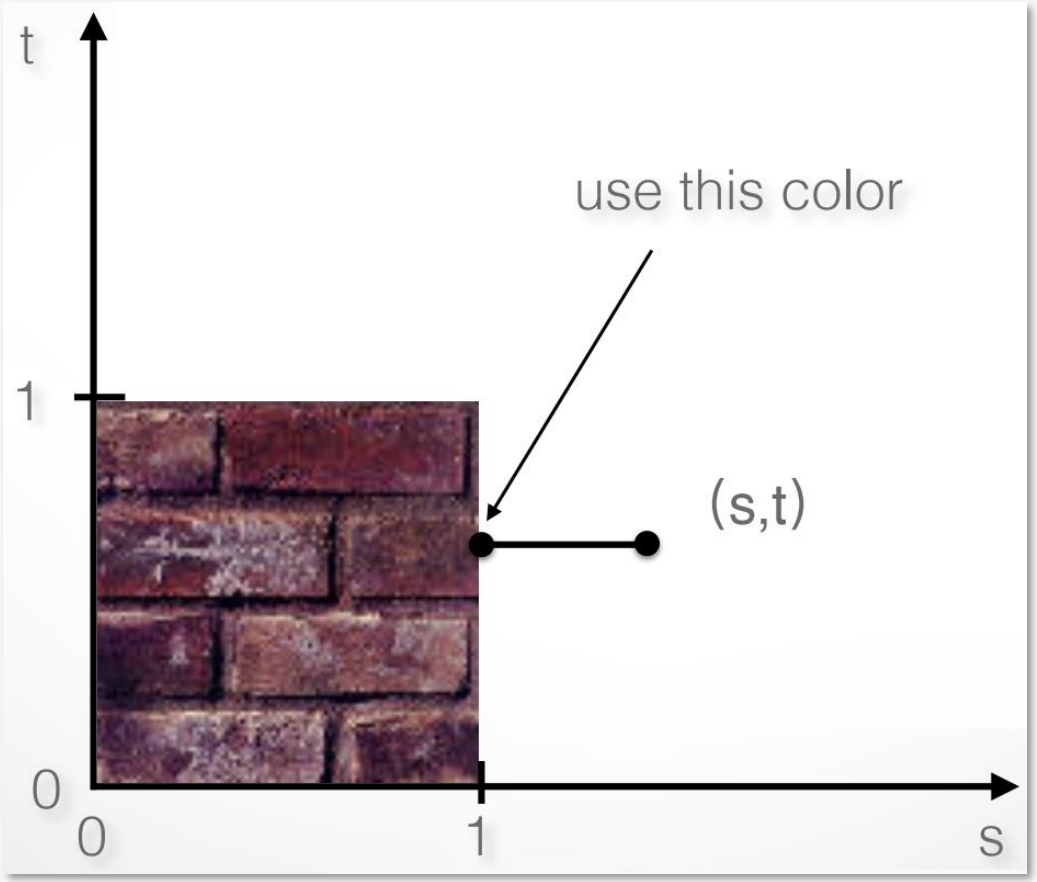
Solution 1: Repeat Texture

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

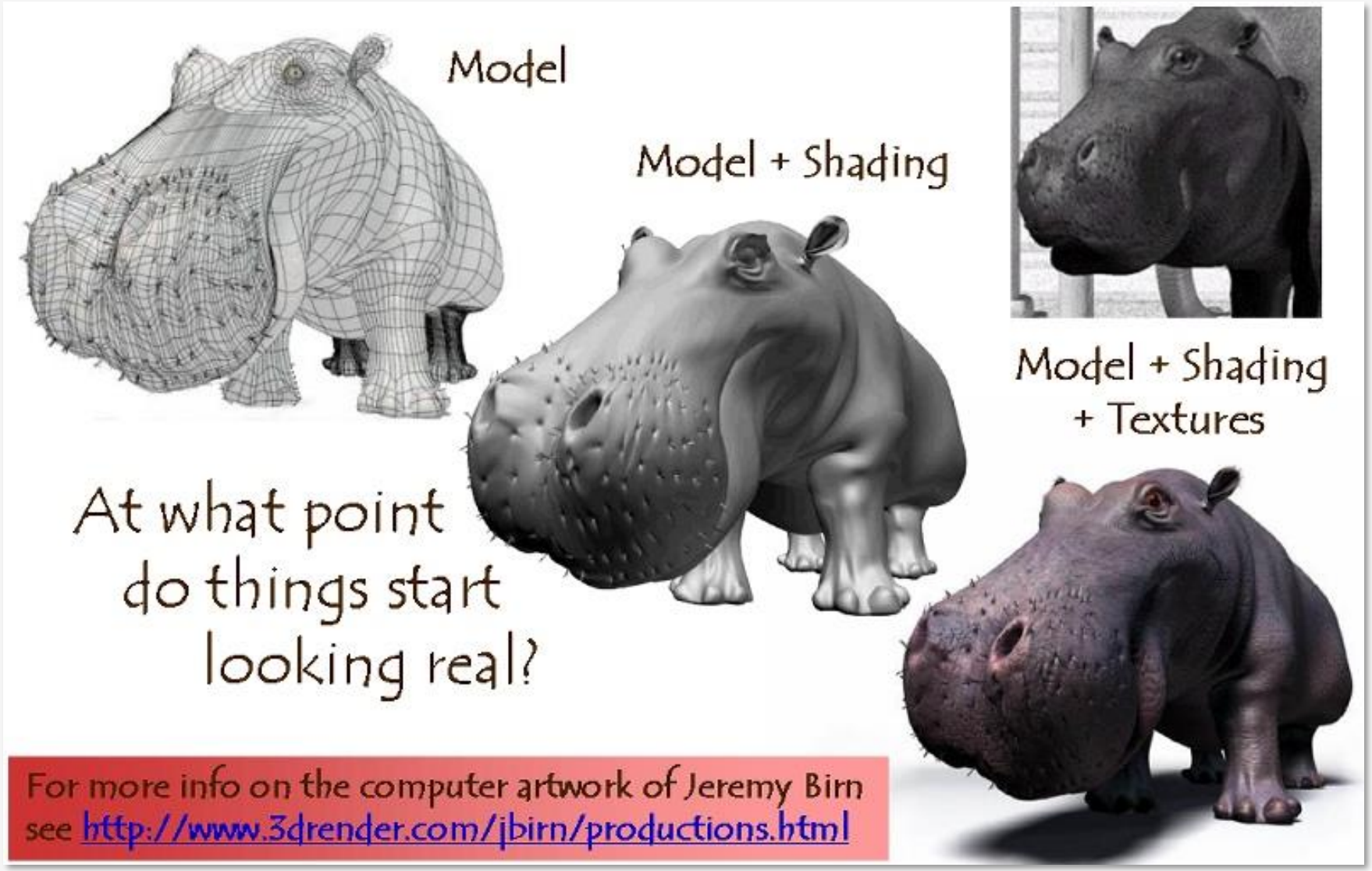


Solution 2: Clamp to [0, 1]

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



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 in OpenGL

- During your initialization:
  1. Read texture image from file into an array in memory, or generate the image using your program
  2. Specify texture mapping parameters
     - wrapping, filtering, etc.
  3. Initialize and activate the texture
- In refresh(window):
  1. Enable OpenGL texture mapping
  2. Draw objects: Assign texture coordinates to vertices
  3. Disable OpenGL texture mapping

Texture Objects

- Like display lists for texture images
  - one image per texture object
  - may be shared by several graphics contexts
- Generate texture names (IDs)

```
tex_id = glGenTextures( n )
```

- Bind textures before using

```
glBindTexture( target, tex_id )
```

- To unbind textures

```
glBindTexture( target, 0 )
```

Initializing the Texture

- Do once during initialization, for each texture image in the scene, by calling `glTexImage2D`
- The dimensions of texture images must be powers of 2
  - if not, rescale image or pad with zero
  - or can use OpenGL extensions
- Can load textures dynamically if GPU memory is scarce

glTexImage2D

```
glTexImage2D(GL_TEXTURE_2D, level, internalFormat, w, h, border, format, type, data)
```

- `GL_TEXTURE_2D`: specifies that it is a 2D texture
- `level`: used for specifying levels of detail for mipmapping (default: 0)
- `internalFormat`
  - often: GL_RGB or GL_RGBA
  - determines how the texture is stored internally
- `w`, `h` are width and height of the data
  - The size of the texture must be powers of 2
- `border` (often set to 0)
- `format`, `type`
  - specifies what the input data is (GL_RGB, GL_RGBA, …)
  - specifies the input data type (GL_UNSIGNED_BYTE, GL_BYTE, …)
  - regardless of `format` and `type`, OpenGL converts the data to `internalFormat`
- data: pointer to the image buffer

Enable/Disable texture mode

- Must be done before rendering any primitives that are to be texture-mapped

  `glEnable(GL_TEXTURE_2D)`

  `glDisable(GL_TEXTURE_2D)`

- Successively enable/disable texture mode to switch between drawing textured/non-textured polygons

- Changing textures:
  - in standard OpenGL without extensions, only one texture is active at any given time
  - with OpenGL extensions, more than one can be used simultaneously; this is called *multitexturing*

- Use `glBindTexture` to select the active texture

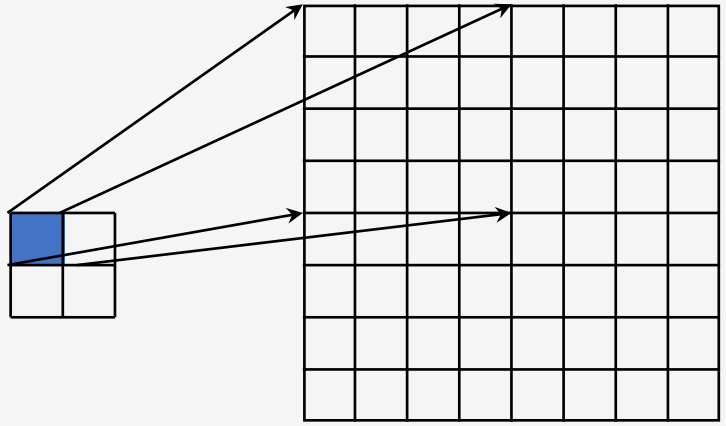
Filtering and Mipmaps

Filter Mode

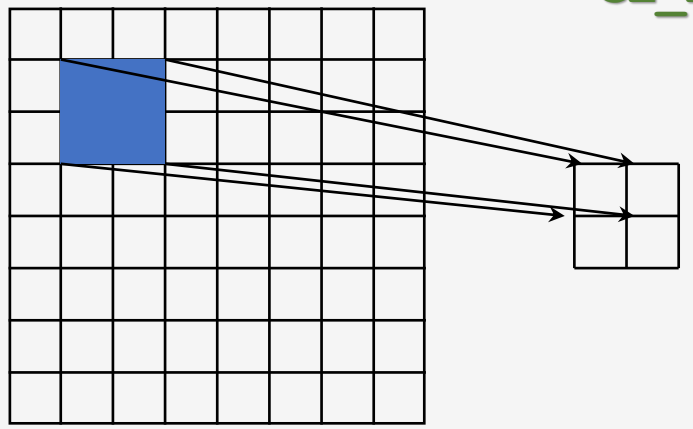
```
glTexParameteri( GL_TEXTURE_2D, type, mode )
```

GL_TEXTURE_MAG_FILTER GL_NEAREST, GL_LINEAR

GL_TEXTURE_MIN_FILTER GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST GL_NEAREST_MIPMAP_LINEAR GL_LINEAR_MIPMAP_NEAREST GL_LINEAR_MIPMAP_LINEAR

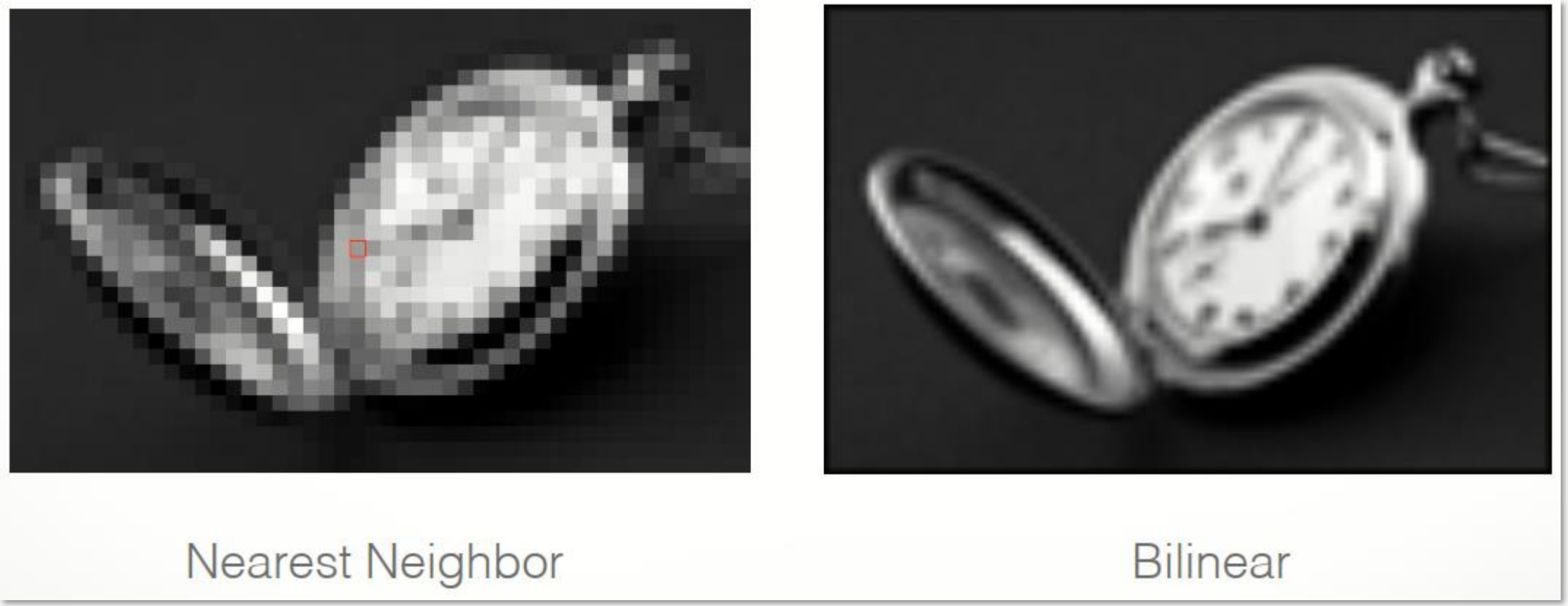


Magnification

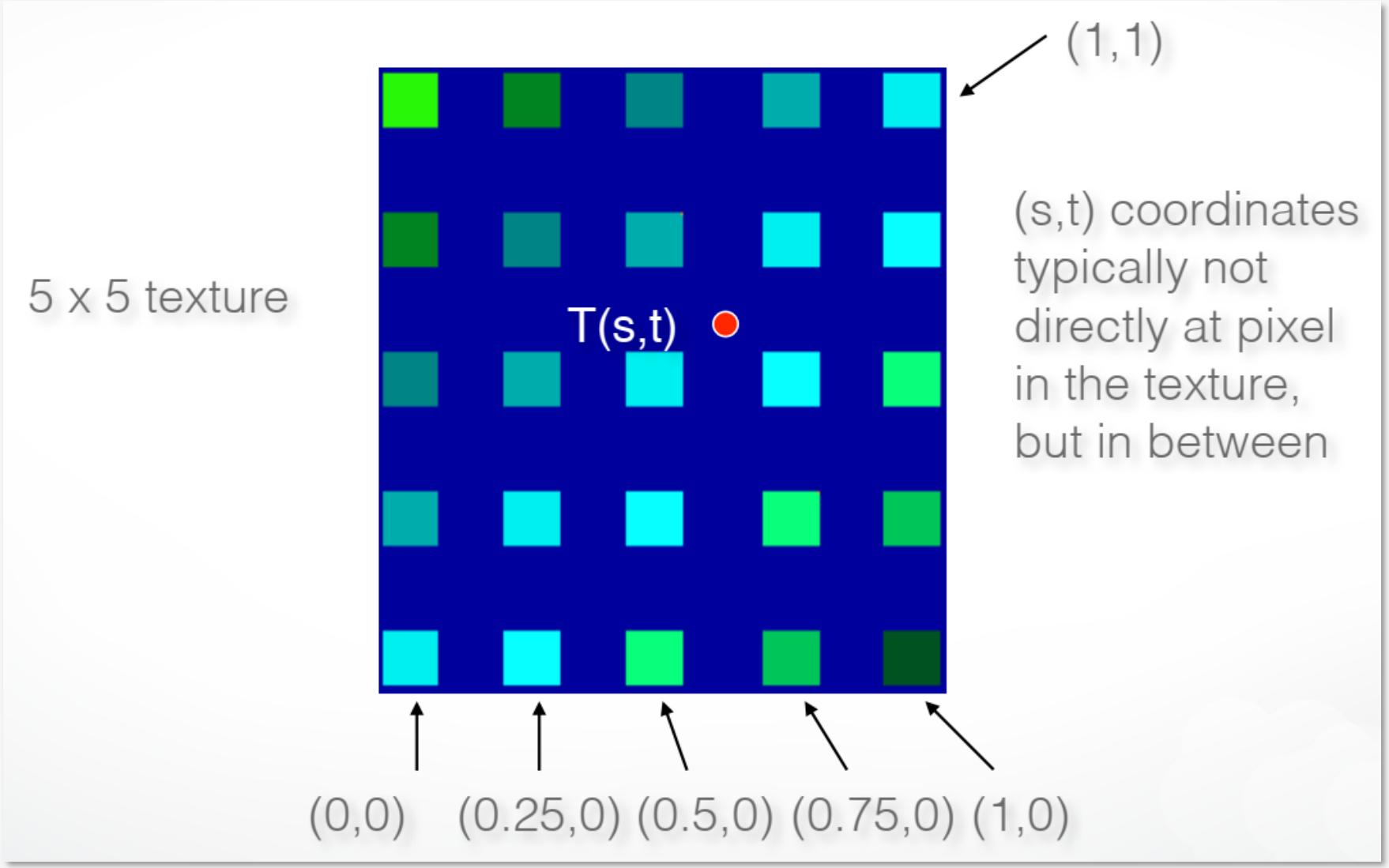


Minification

Nearest Neighbor vs. Bilinear Interpolation



Texture Interpolation



Texture Interpolation in OpenGL

- (s,t) coordinates typically not directly at pixel in the texture, but in between
- Solutions:
  - Use the nearest neighbor to determine color
    - faster, but worse quality

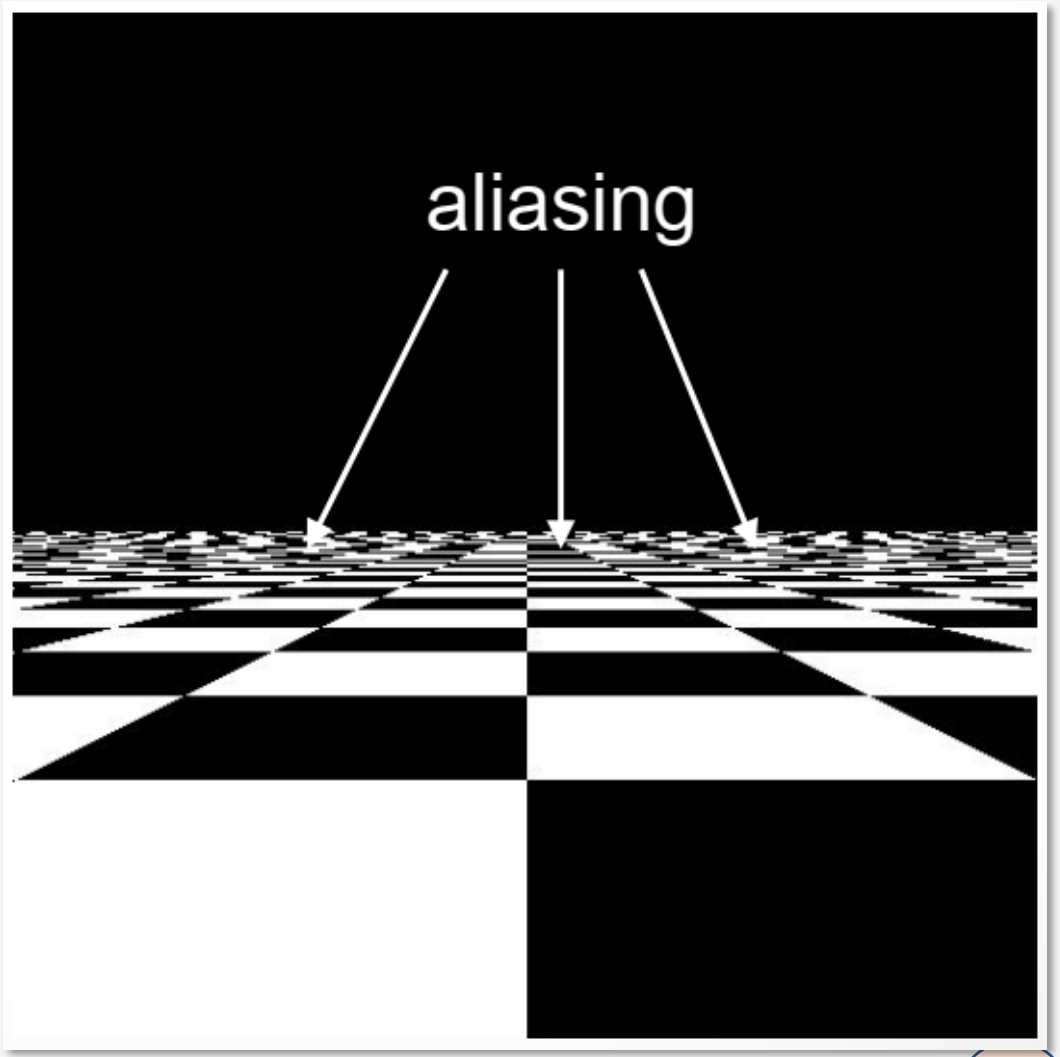
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
```

  - Linear interpolation
    - incorporate colors of several neighbors to determine color
    - slower, better quality

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

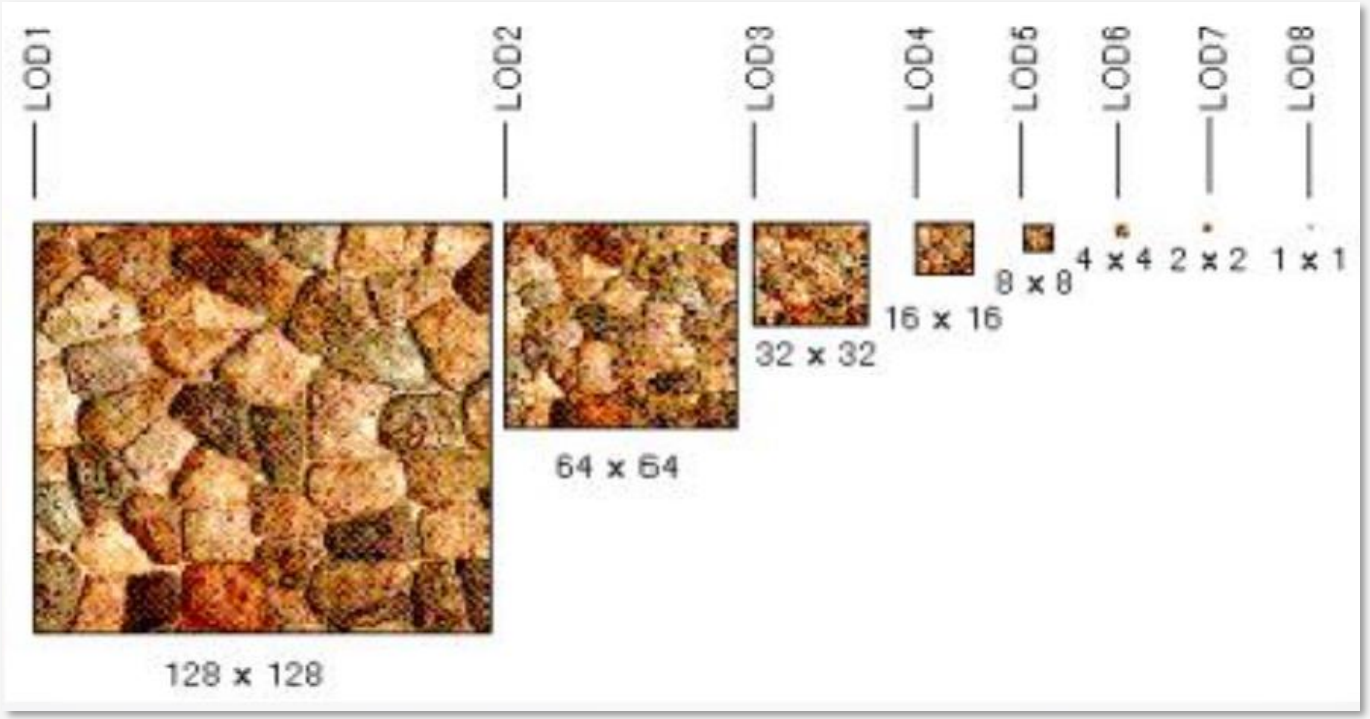
Filtering

- Texture image is shrunk in distant parts of the image
- This leads to aliasing
- Can be fixed with filtering
  - bilinear in space
  - trilinear in space and level of detail (mipmapping)



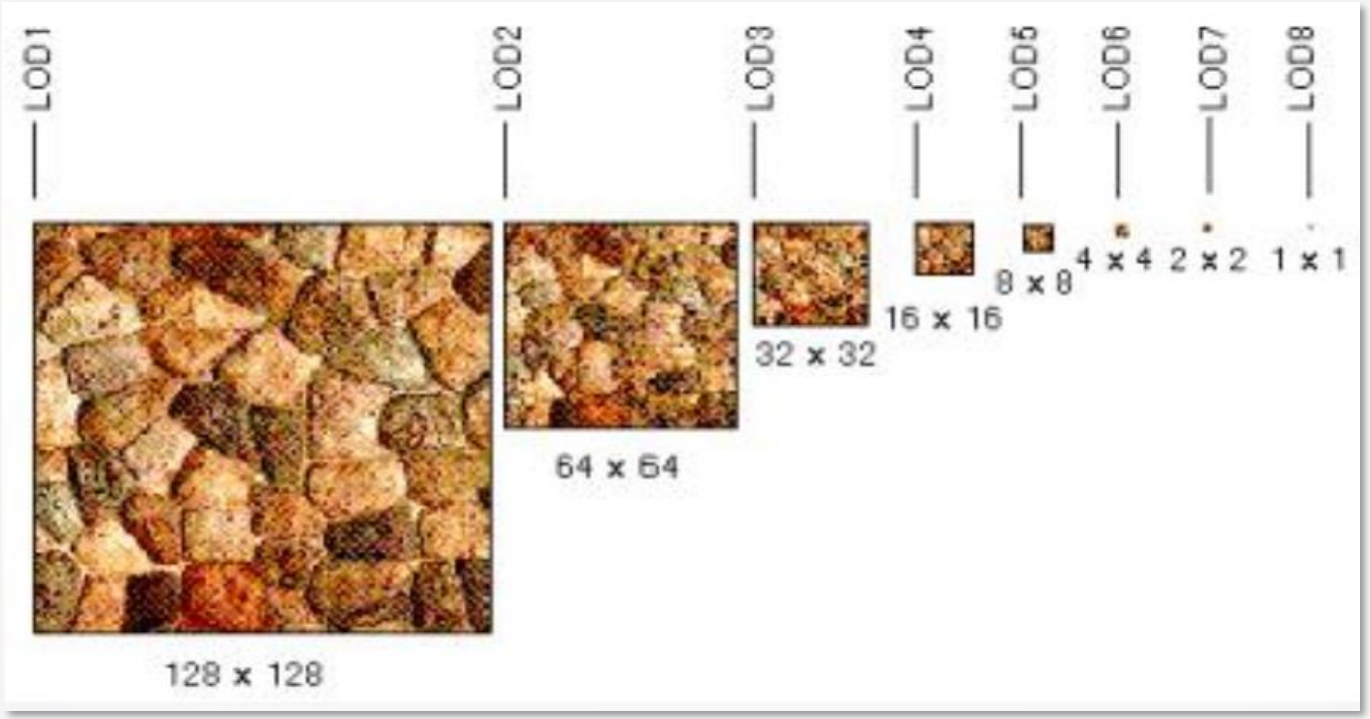
Mipmapping

- Pre-calculate how the texture should look at various distances, then use the appropriate texture at each distance
- Reduces or fixes the aliasing problem



Mipmapping

- Each mipmap (each image below) represents a level of depth (LOD)
- Powers of 2 make things much easier



Mipmapping in OpenGL

```
gluBuild2DMipmaps(GL_TEXTURE_2D, components, w, h, format, type, data)
```

- this will generate all the mipmaps automatically
- or call `glGenerateMipmap(GL_TEXTURE_2D)` after `glTexImage2D(…)`

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, mode)
```

- This will tell GL to use the mipmaps for the texture
- mode can be a value of the following:

```
GL_NEAREST_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MIPMAP_NEAREST
GL_LINEAR_MIPMAP_LINEAR
```

Example of Code

Example of Code

draw

```
glEnable(GL_TEXTURE_2D)
glBegin(GL_QUADS)
glTexCoord2f(0.0, 1.0); glVertex3f(-4.0, 0.8,-8.0)
glTexCoord2f(0.0, 0.0); glVertex3f(-4.0,-0.5, 4.0)
glTexCoord2f(1.0, 0.0); glVertex3f( 4.0,-0.5, 4.0)
glTexCoord2f(1.0, 1.0); glVertex3f( 4.0, 0.8,-8.0)
glEnd()
glDisable(GL_TEXTURE_2D)

glfwSwapBuffers(window)
```

initialize

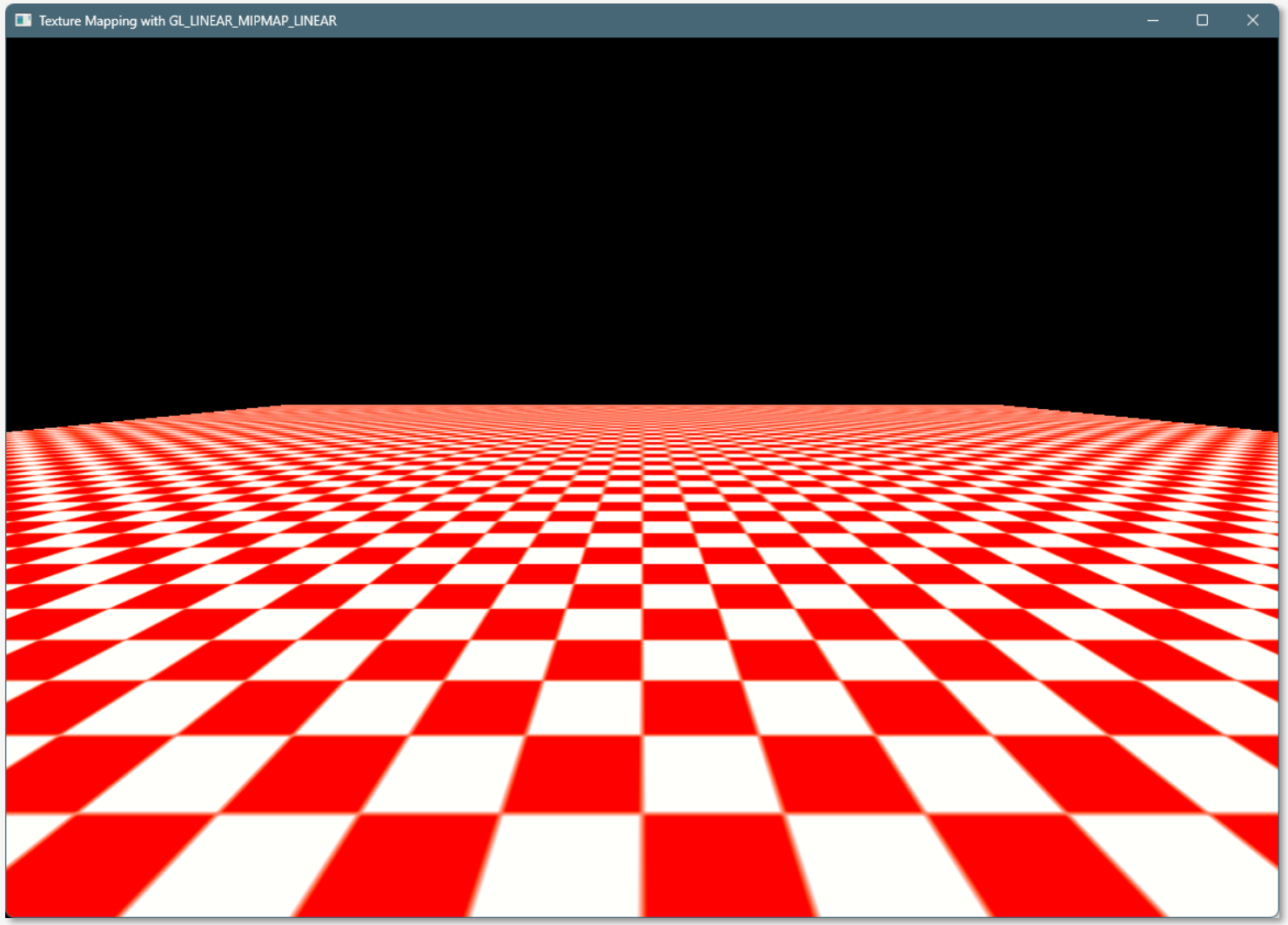
```
tex_id = glGenTextures(1)

glActiveTexture(GL_TEXTURE0)
glBindTexture(GL_TEXTURE_2D, tex_id)
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, texels)
glGenerateMipmap(GL_TEXTURE_2D)

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
```

Demo

Mipmapping Demo



References

Angel, E., Shreiner, D., & Shreiner, V. (2007). *An interactive introduction to OpenGL programming. In ACM SIGGRAPH 2007 courses (pp. 1-124).*

CSCI 420: Computer Graphics FS 2018 by Prof. Dr. Hao Li