Texture Mapping

CS 418: Interactive Computer Graphics

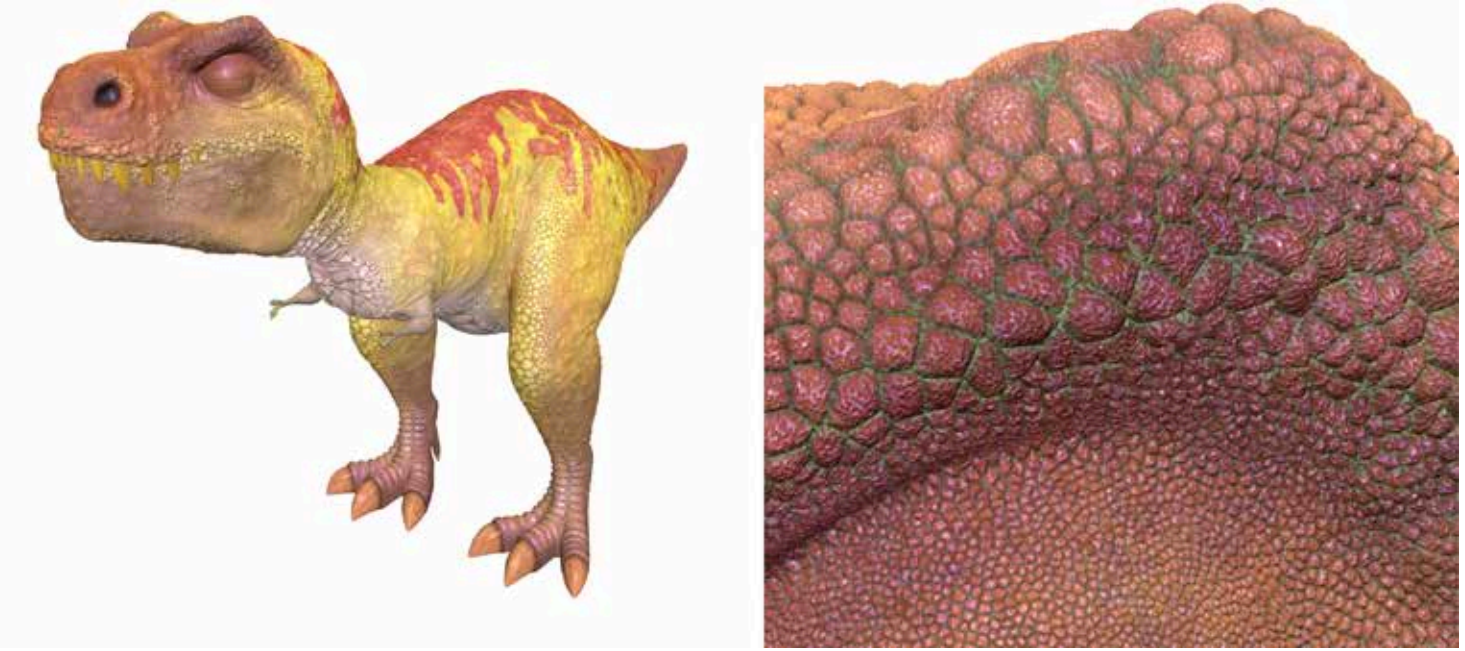
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

"All it takes is for the rendered image to look right."
—Jim Blinn

Eric Shaffer

The Limits of Geometric Modeling

- Graphics cards can render over 20 billion polygons per second
  (NVIDIA 2080 TI in 2018)
- That number is still insufficient for many visual phenomena
- Consider rendering a herd of 100s of bumpy-skinned dinosaurs



Or Consider Modeling an Orange

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

Modeling an Orange

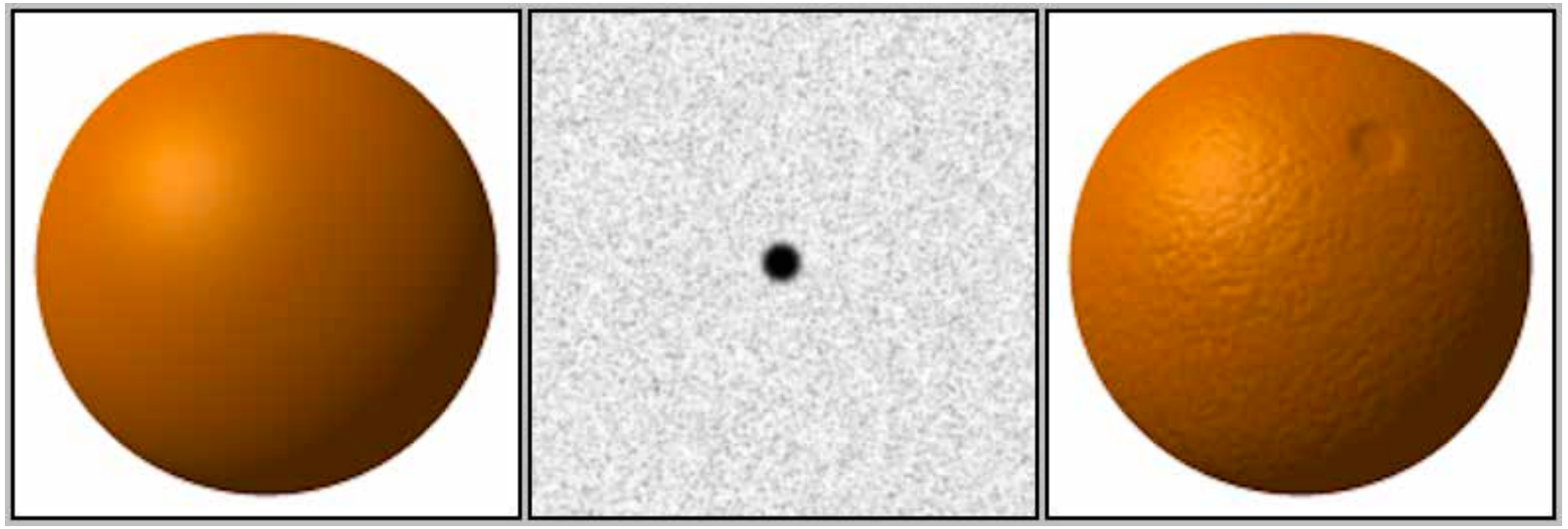
- Take a picture of a real orange
- “paste” pixels of the image onto simple geometric model
  - This process is known as texture mapping
  - Specifically *image texturing*
- Still might be problematic…
  - Looking at the orange in a rendered scene
  - How could you tell the colors on the orange aren’t generated by shading?

Modeling an Orange

Another alternative would be ***bump mapping***

Use an image that specifies the normal to use to shade the surface

- This way, can render an “bumpy” surface during shading
- Without modeling the bumpy surface with lots of triangles



Some Types of Texture Mapping

- Image Texturing
  - Uses images to fill inside of polygons
- Environment Mapping
  - Uses a picture of the environment for texture maps
  - Allows simulation of mirror-like surfaces
- Bump mapping
  - Alters normal vectors during the rendering process
  - Generates a bumpy looking surface

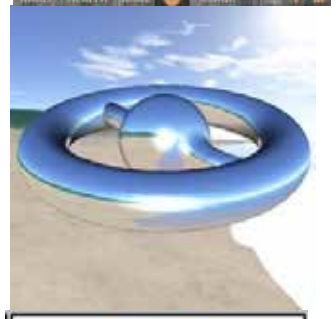
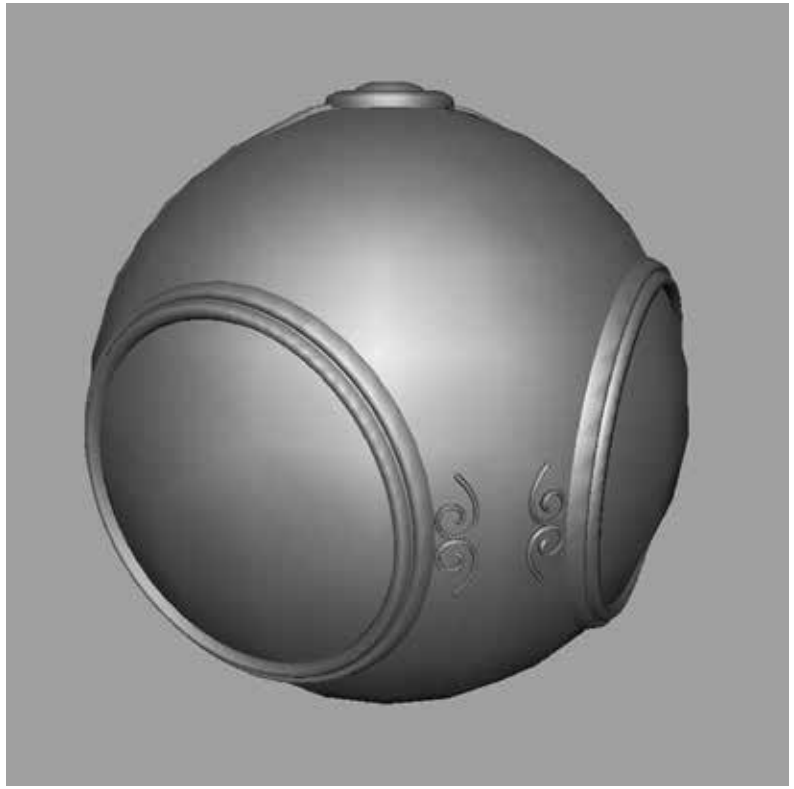


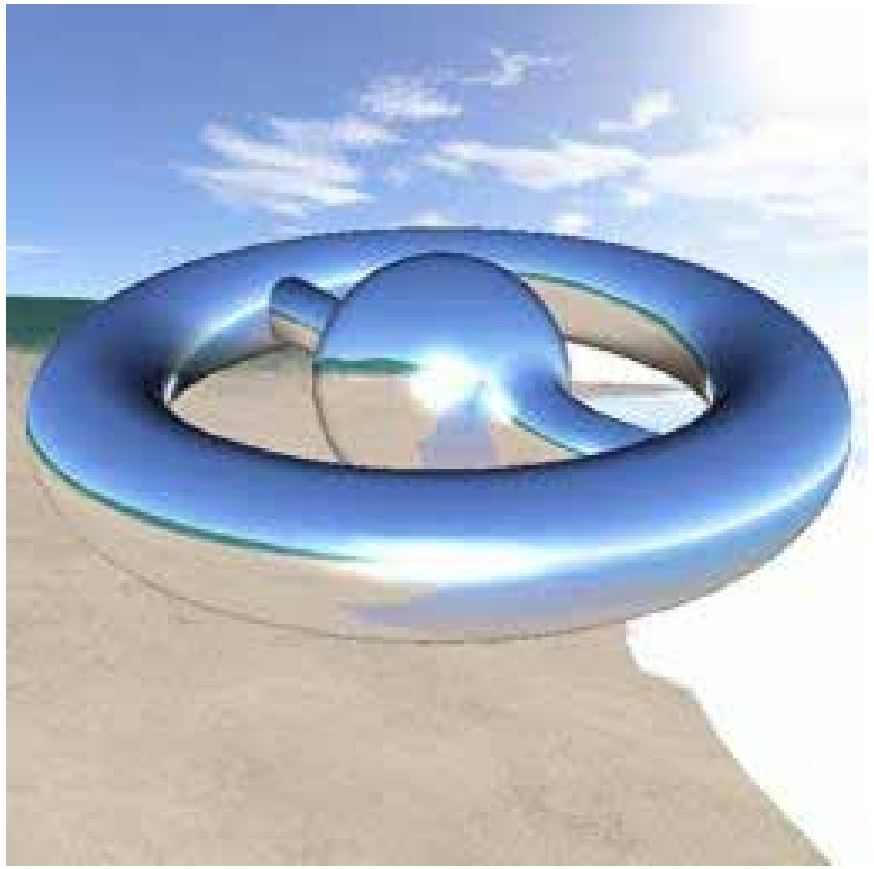
Image Texturing



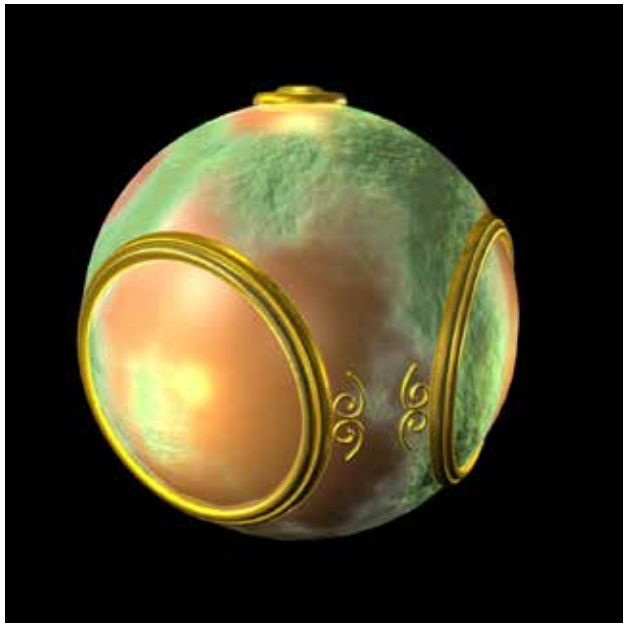
geometric model

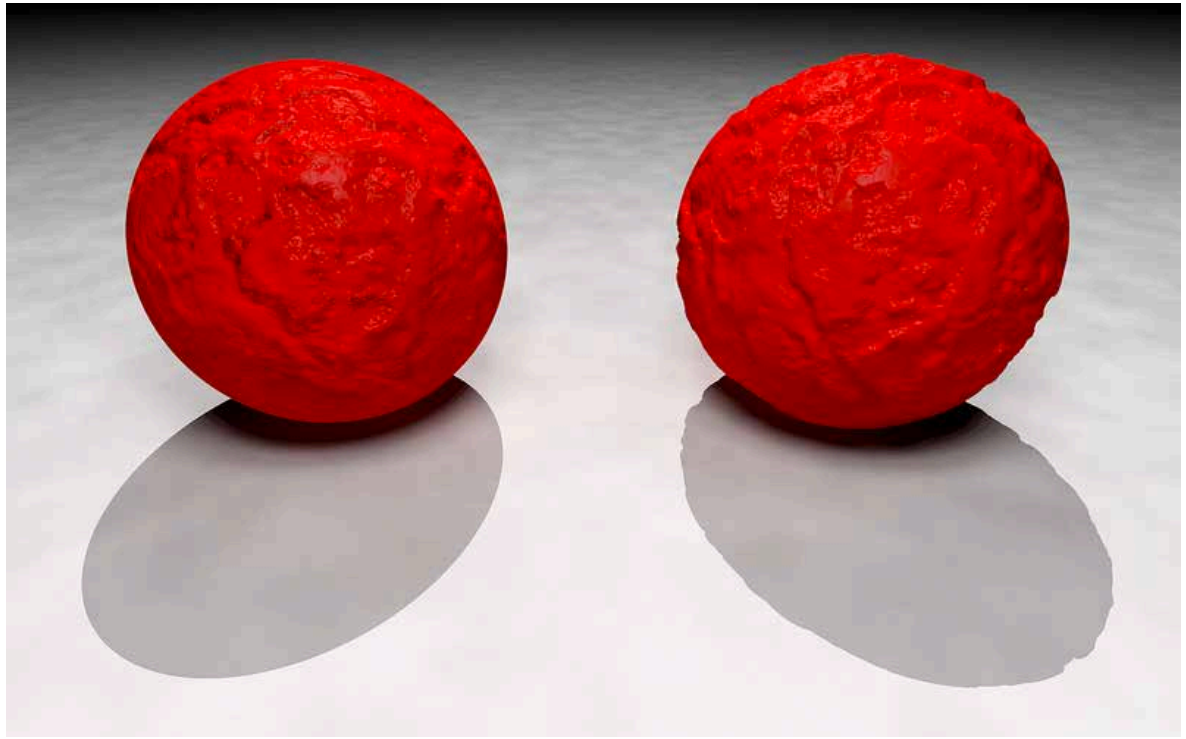
texture mapped

Environment Mapping



Bump Mapping



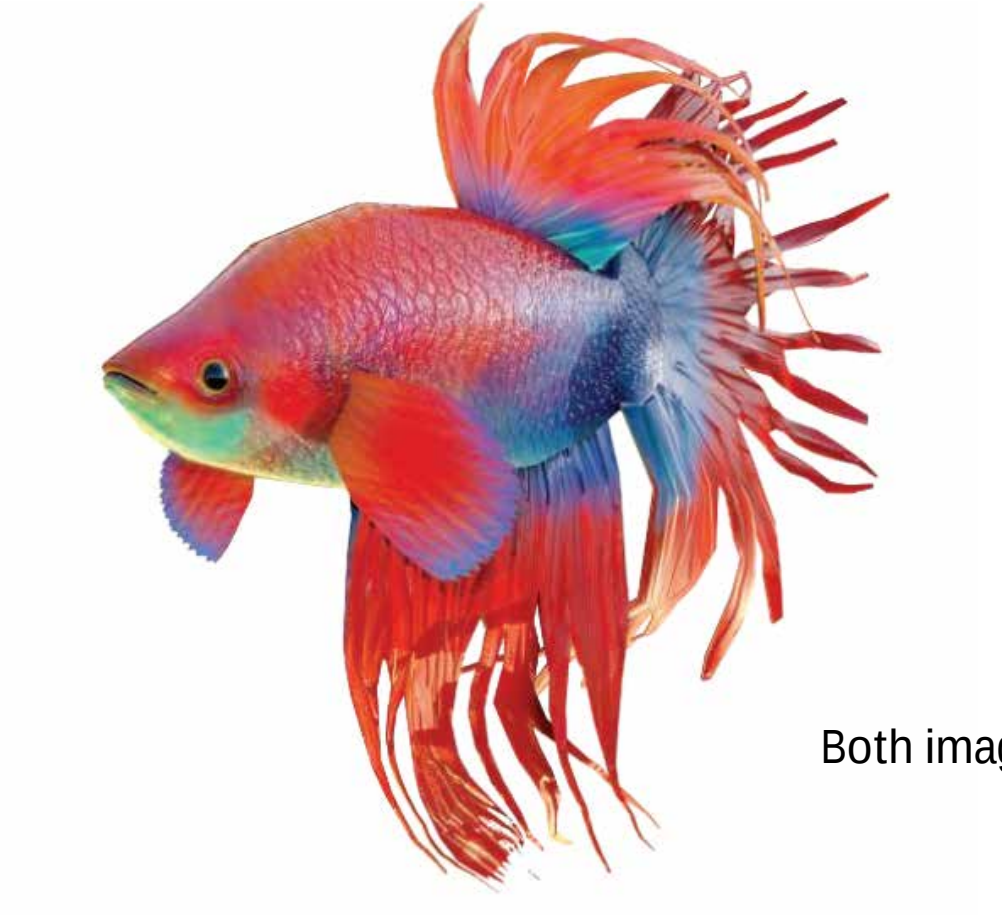


So What is Texture?

A surface's texture is its look and feel—just think of the texture of an oil painting. In computer graphics, texturing is a process that takes a surface and modifies its appearance at each location using some image, function, or other data source. As an example, instead of precisely representing the geometry of a brick wall, a color image of a brick wall is applied to a rectangle, consisting of two triangles. When the rectangle is viewed, the color image appears where the rectangle is located. Unless the viewer gets close to the wall, the lack of geometric detail will not be noticeable.

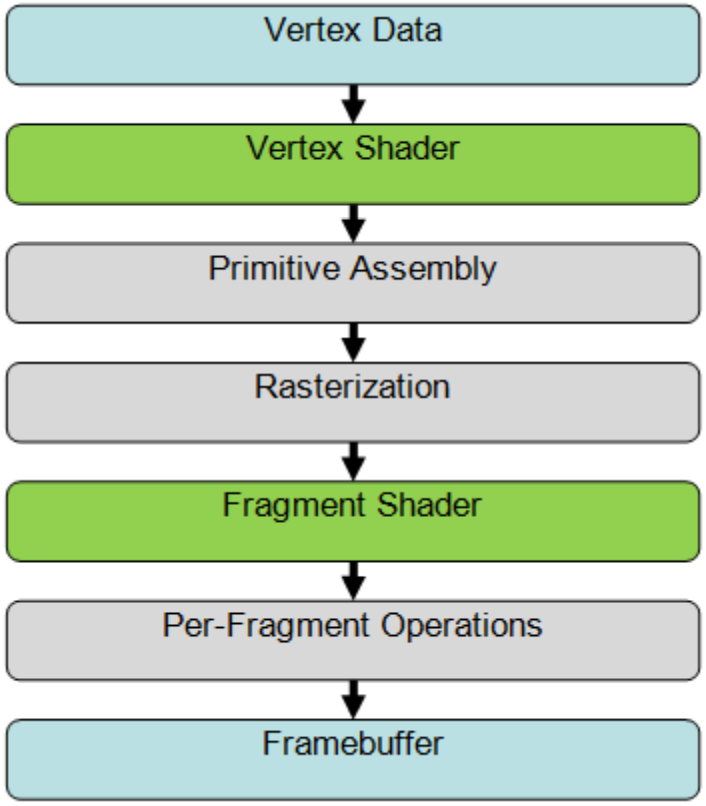
-- Akenine-Moeller, Tomas; Haines, Eric; Hoffman, Naty. Real-Time Rendering, Fourth Edition

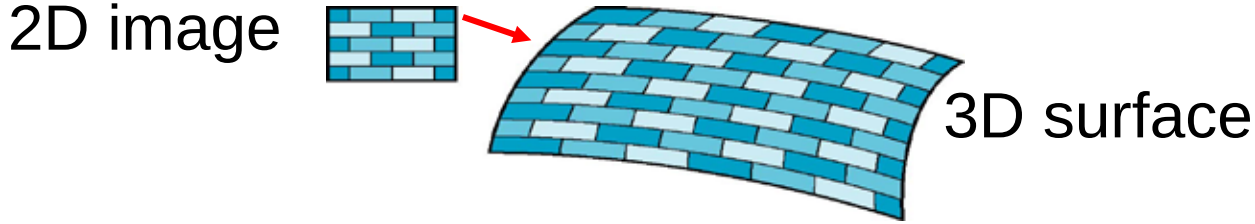
For example



Both image texturing and bump mapping are used here

Texturing Mapping and the Pipeline





In the simplest form texture mapping is a process in which

- a fragment on a surface to be rendered
- is mapped to a color in a texture (an image)
- and then this color (called a texel) is used in shading

Mapping is implemented in the fragment shader

- Efficient because few polygons make it past the clipper

Texture Mapping

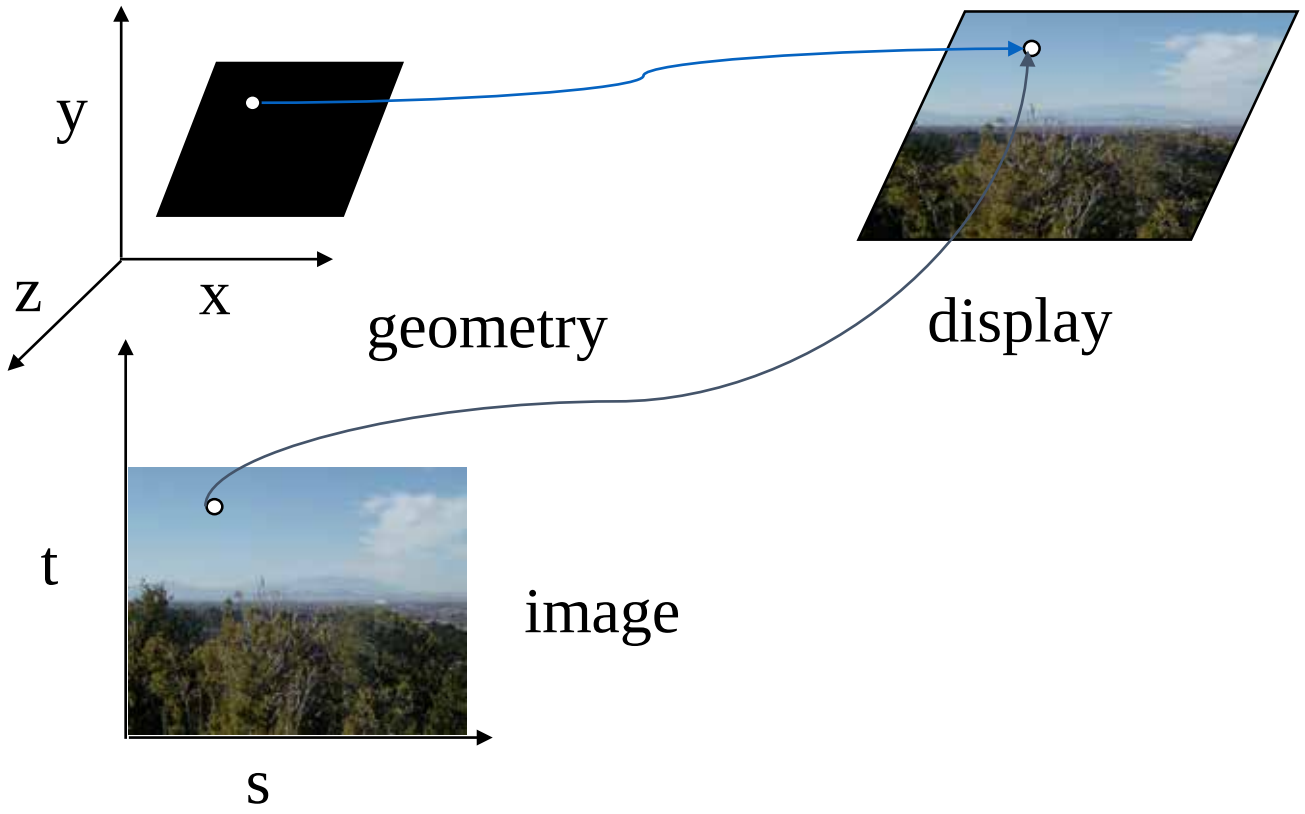
Mapping means we need a function

- Given a point on a surface (a fragment)
- We want to know to which texel in the texture it corresponds to

Need a map of the form

s = s(x,y,z)

t = t(x,y,z)



We will use the word **texel** to indicate a pixel in the texture image…so

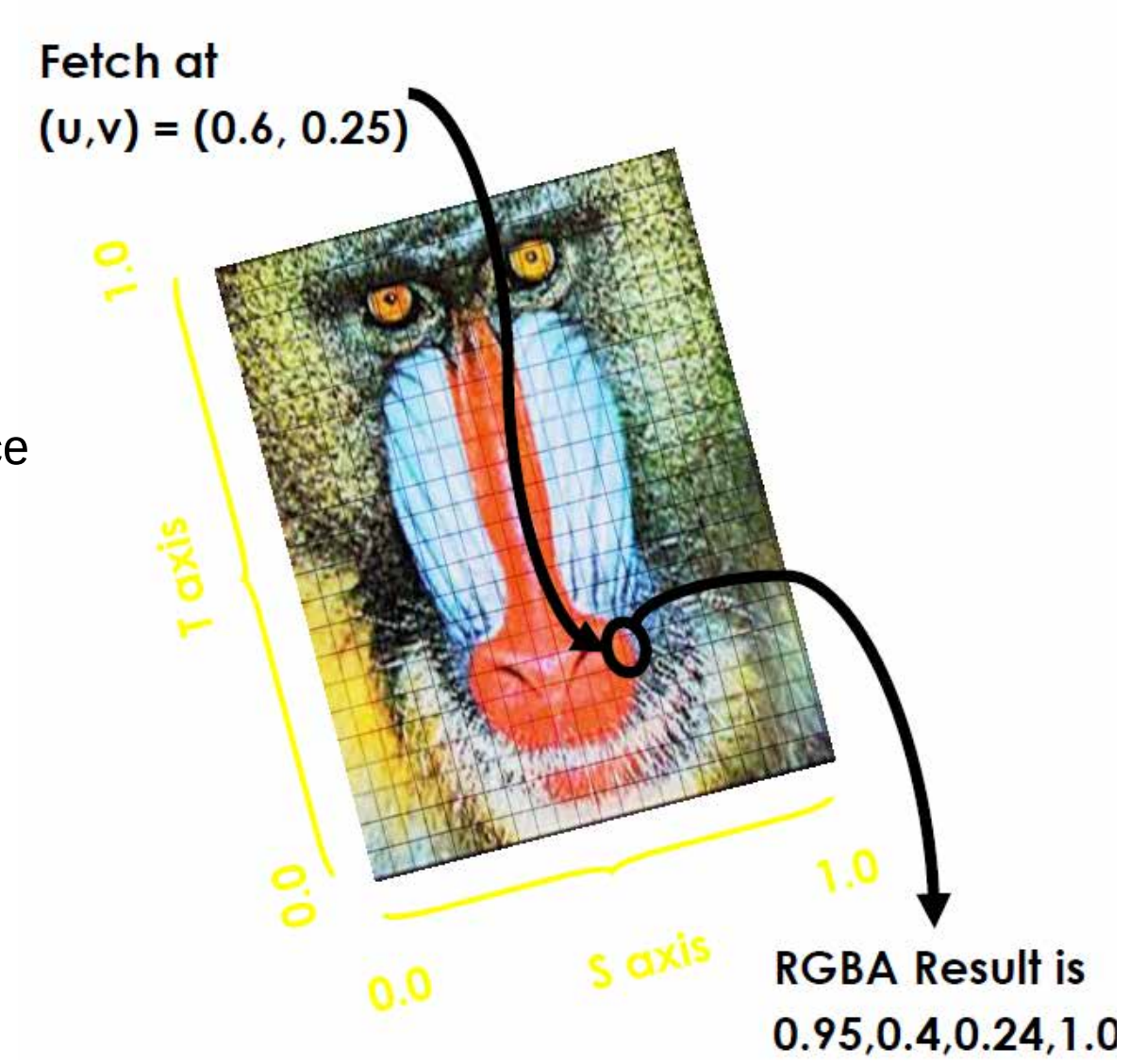
Specifying a Texture Mapping Function

- You need to come up with a mapping from

  surface vertices *texture space*

- Specify mapping using (u,v) vertex attributes
- The (u,v) coordinates map into a parametric texture space

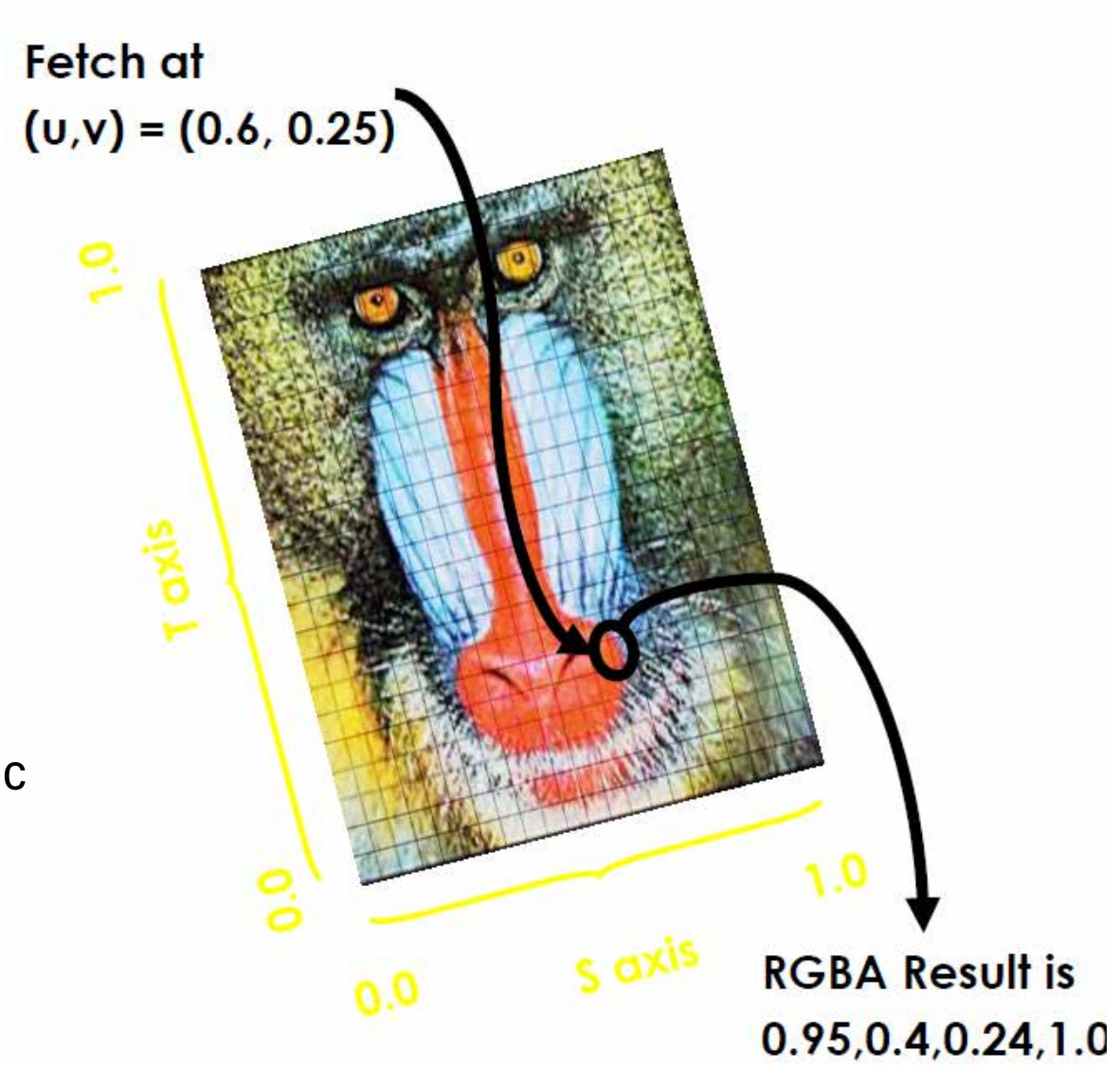
- Generally (u,v) is in [0,1] x [0, 1]
  - We can allow coordinates to go outside that range…
  - if we tell WebGL how to handle that event
    …more on that later



A Word About Coordinates

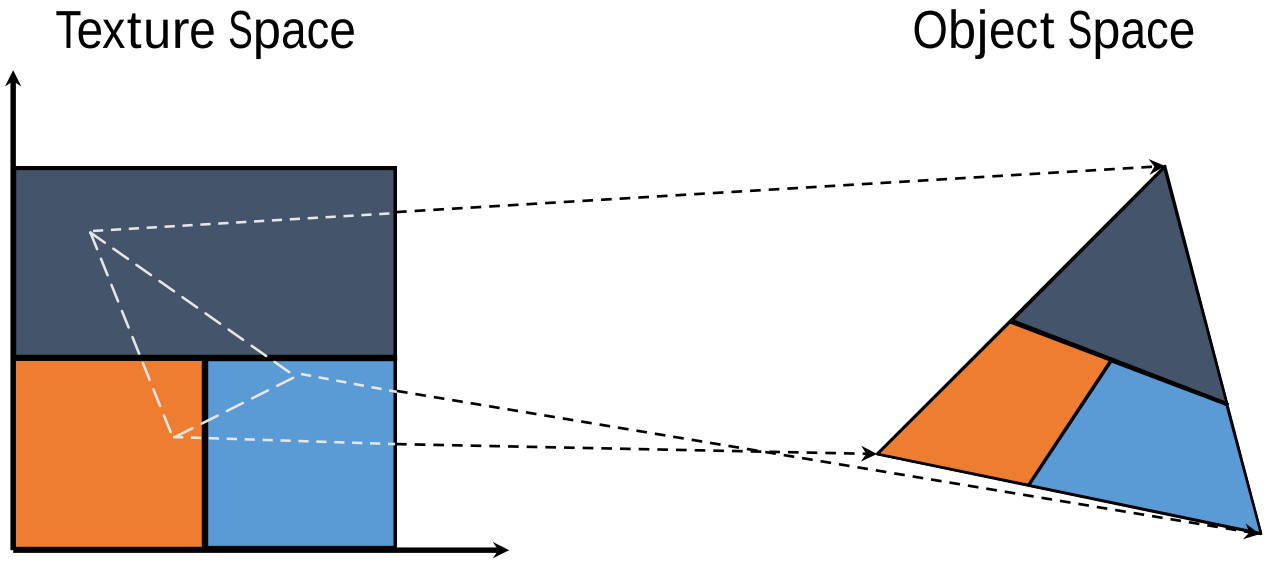
We will use the following conventions

- (u,v) are the texture coordinates in the parametric space
  - Typically in [0,1]x[0,1]
- (s,t) are the texel coordinates
  - For example in a 32x32 image (s,t) would be in [0,31]x[0,31]
  - (s,t) are floating point coordinates…
  - …we may not map directly onto an integer texel location
  - …and then we need to figure out how to choose the best texel
- Some websites/books/posts use (s,t) to denote parametric coordinates…so be aware….



Mapping a Texture

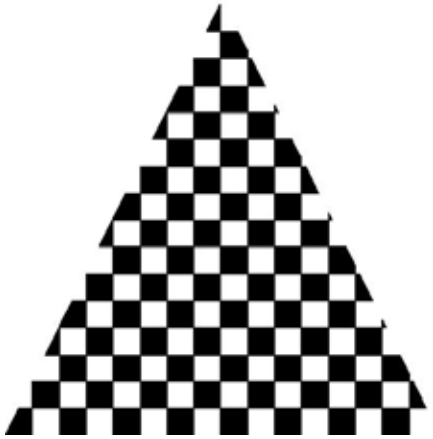
- Based on parametric texture coordinates
- Specify as a 2D vertex attribute



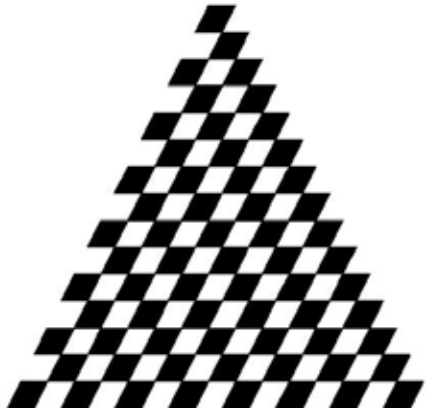
Mapping a Texture

Can be distortions

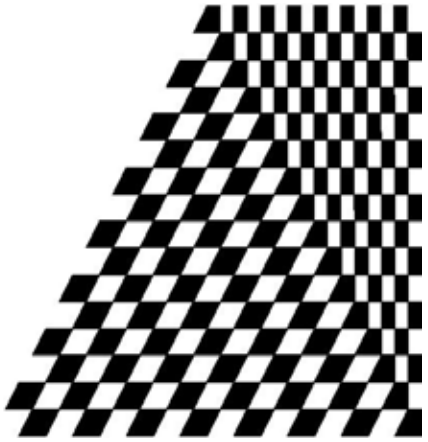
good selection
of tex coordinates



poor selection
of tex coordinates



texture stretched
over trapezoid
showing effects of
bilinear interpolation



Basic Steps in WebGL

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
- Use an image in a standard format such as JPEG
  - Scanned image
  - Generate by application code
- WebGL supports only 2 dimensional texture maps
  - no need to enable as in desktop OpenGL
  - desktop OpenGL supports 1-4 dimensional texture maps

Define Image as a Texture

type of texture, e.g.

used for mipmapping (discussed later)

elements per texel

width and height of in pixels

used for smoothing (discussed later)

describe texels

pointer to texel array

Example

- Once the image data is loaded into an array
  - We'll discuss how to do that in a future lecture
- You then need to tell WebGL the image is a texture
- WebGL has a concept of the *current* texture
  - gl.bindTexture sets the current texture
  - this is the texture that gets operated on

```
function handleLoadedTexture(texture) {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
                    gl.UNSIGNED_BYTE, texture.image);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
      gl.bindTexture(gl.TEXTURE_2D, null);
  }
```

Example

- When the image data is given to WebGL you tell it how to “unpack” it

- Here, we tell it to flip the image vertically
  - Most image formats increase coords going down vertical axis
  - WebGL expects the coordinates to increase going up

```
function handleLoadedTexture(texture) {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
                    gl.UNSIGNED_BYTE, texture.image);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
      gl.bindTexture(gl.TEXTURE_2D, null);
   }
```

Example

- We can give WebGL hints about how to scale the image data
- Need to magnify if we have lots of fragments and few texels
- Need to minify if we have few fragments and lots of texels

```
function handleLoadedTexture(texture) {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
                          gl.UNSIGNED_BYTE, texture.image);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
      gl.bindTexture(gl.TEXTURE_2D, null);
   }
```

Examples

Nearest neighbor filtering

Bilinear Interpolation

Magnification: Nearest Neighbor

Nearest Neighbor Filtering

Sample the texel (s,t) given by:

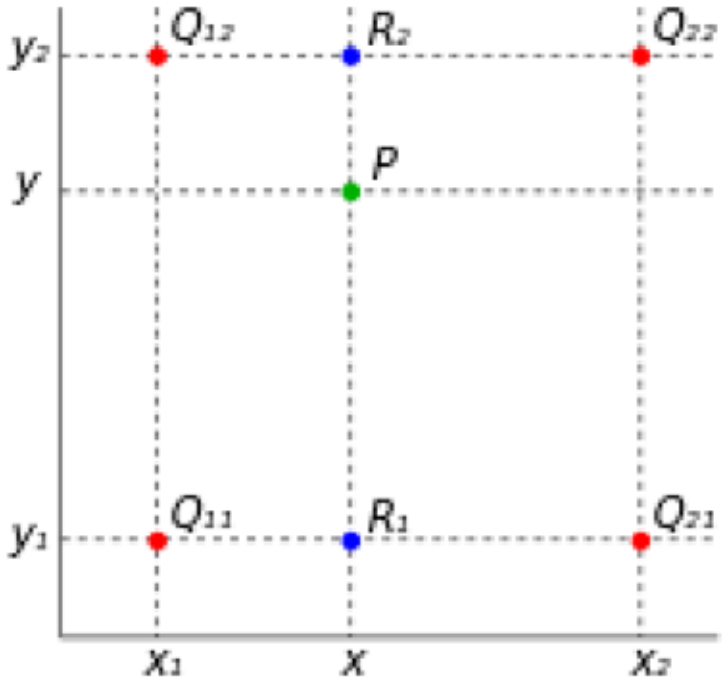
$$s = \text{round}\left((u \times width) - \frac{1}{2}\right)$$

$$t = \text{round}\left((v \times height) - \frac{1}{2}\right)$$

Magnification: Bilinear Interpolation

- In bilinear interpolation, we estimate a value for a function
  - On a 2D grid...with function samples at the grid vertices
- We interpolate first in one direction (e.g. the x direction)
  - Interpolate using linear interpolation twice
  - Find 2 points...one on each edge
- Then interpolate in the other direction (e.g. the y direction)
  - Linear interpolation again
  - Between the two points from the first round of interpolation

Magnification: Bilinear Interpolation



$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$

$$'(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)$$

Bilinear Filtering

In C/C++ style code a function to do bilinear filtering would look like this:

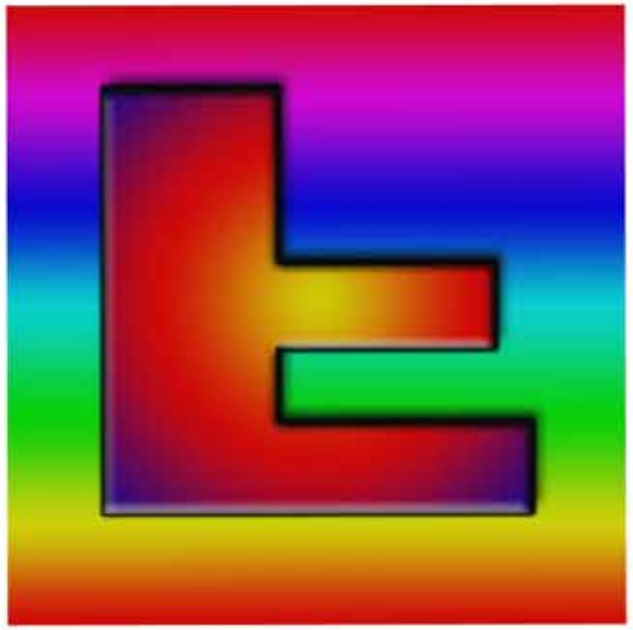
```
Color tex_sample_bilinear(Texture t, float u, float v) {
    u_p = u * t.width - 0.5
    v_p = v * t.height - 0.5
    iu0 = floor(u_p); iu1 = iu0 + 1
    iv0 = floor(v_p); iv1 = iv0 + 1
    a_u = (iu1 - u_p); b_u = 1 - a_u
    a_v = (iv1 - v_p); b_v = 1 - a_v
    return a_u * a_v * t[iu0][iv0] + a_u * b_v * t[iu0][iv1] +
           b_u * a_v * t[iu1][iv0] + b_u * b_v * t[iu1][iv1]
}
```

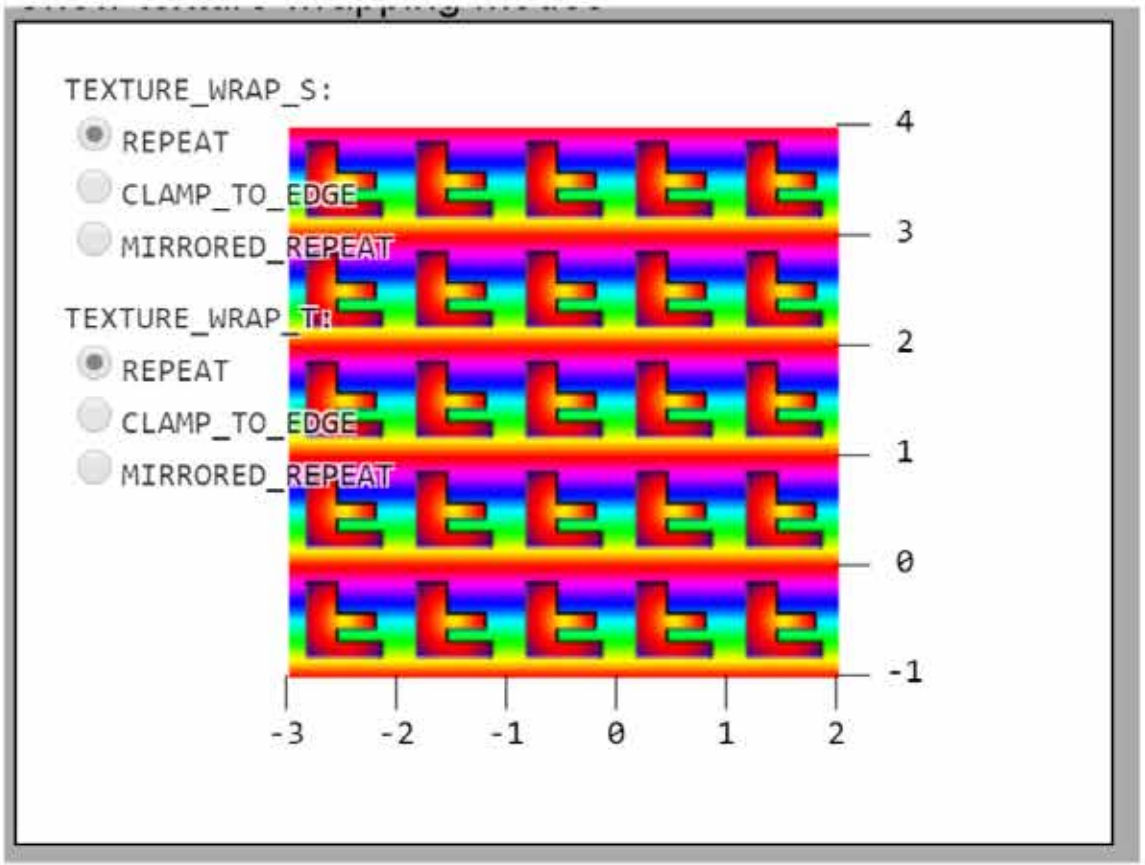
Out of Bound (u,v) Coordinates

- We can let (u,v) coordinates be outside the range [0,1]
  - Yes…they can even be negative
- How these coordinates are handled is defined by the Wrap Mode
- WebGL supports three wrap modes in both the s and t coordinates:
  - REPEAT
  - CLAMP_TO_EDGE
  - MIRRORED_REPEAT
- Example:

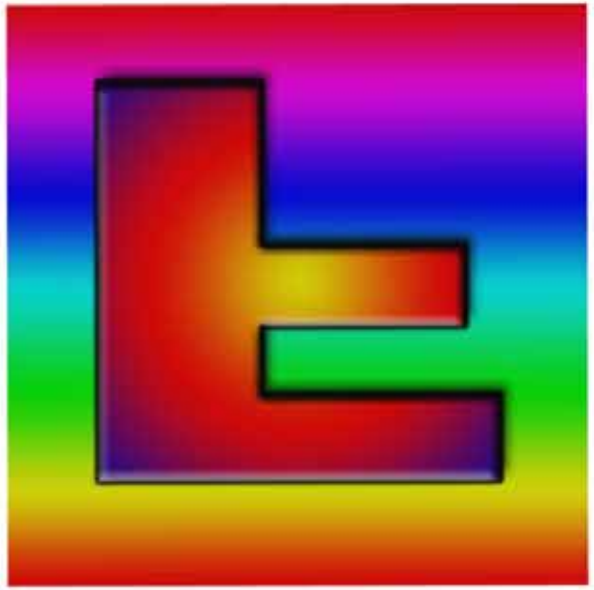
```
// gl.NEAREST is also allowed, instead of gl.LINEAR, as neither mipmap.
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
// Prevents s-coordinate wrapping (repeating).
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
// Prevents t-coordinate wrapping (repeating).
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
```

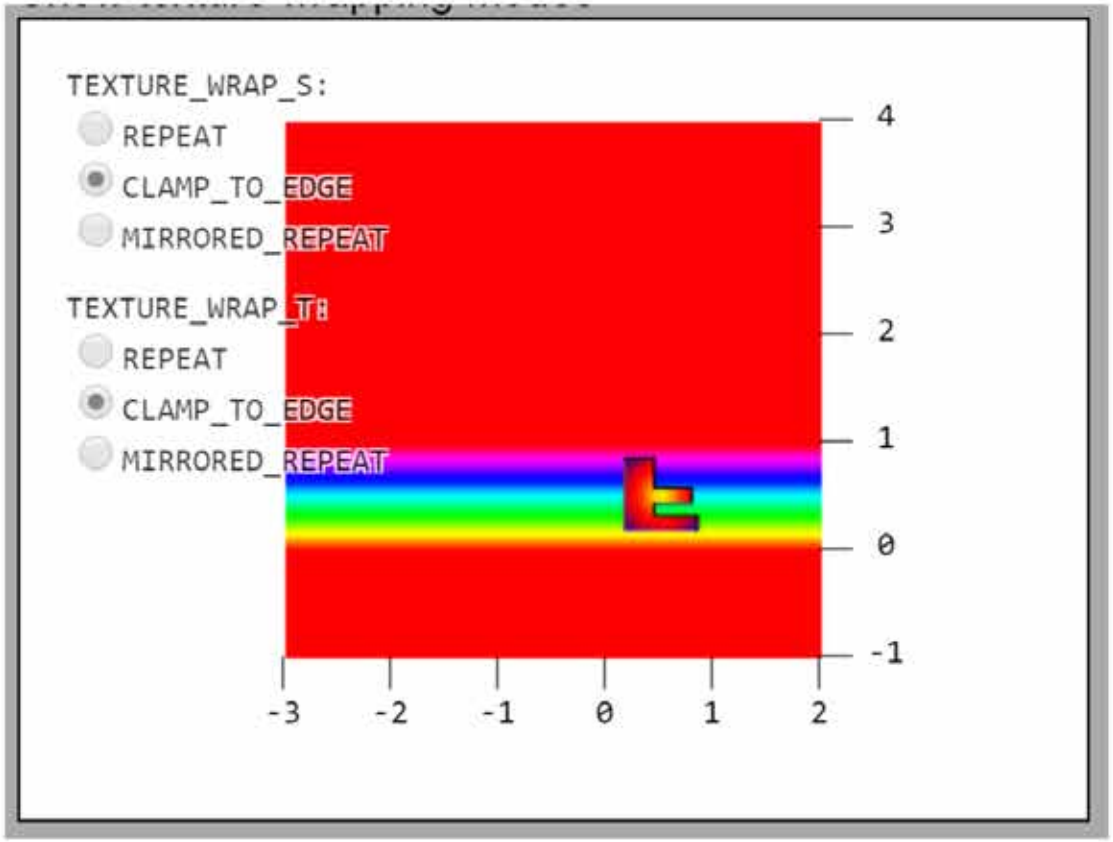
Repeat



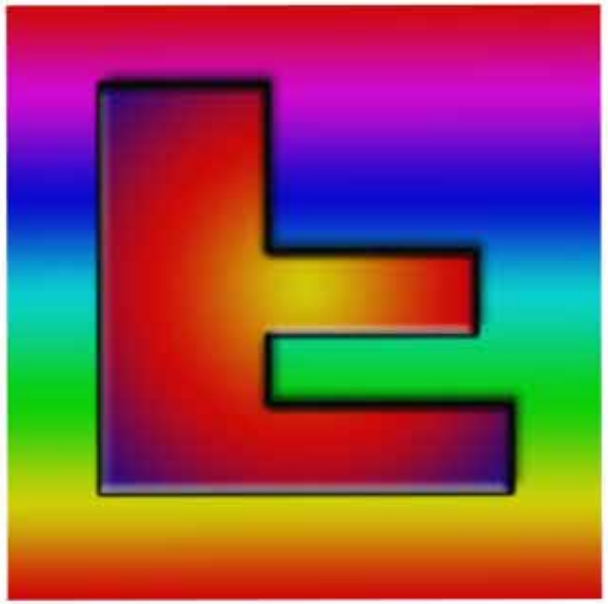

TEXTURE_WRAP_S:
REPEAT
CLAMP_TO_EDGE
MIRRORED_REPEAT
TEXTURE_WRAP_T:
REPEAT
CLAMP_TO_EDGE
MIRRORED_REPEAT
4
3
2
1
0
-1
-3
-2
-1
0
1
2

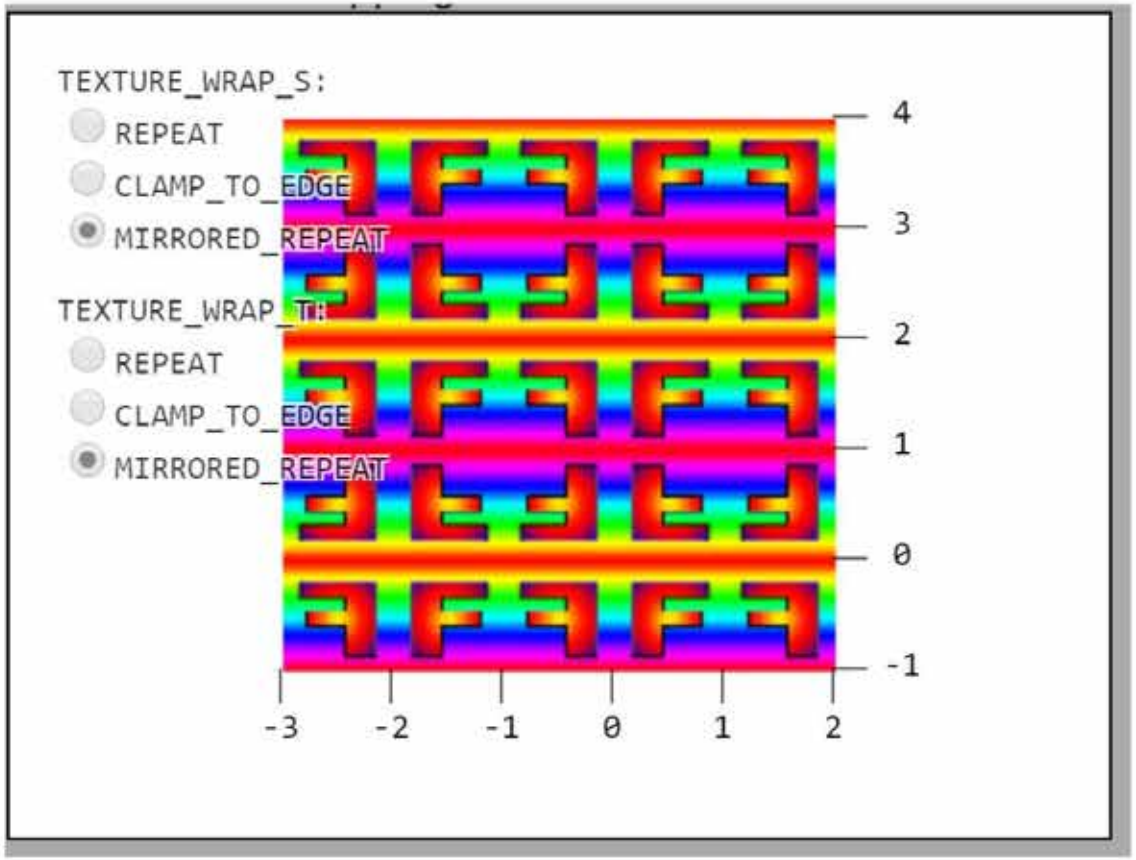
Clamp




TEXTURE_WRAP_S:
REPEAT
CLAMP_TO_EDGE
MIRRORED_REPEAT
TEXTURE_WRAP_T:
REPEAT
CLAMP_TO_EDGE
MIRRORED_REPEAT
4
3
2
1
0
-1
-3
-2
-1
0
1
2

Mirrored Repeat





Sprites versus Textures

Where do you see sprites being used?

Where do you see textures being used?

What is the difference?