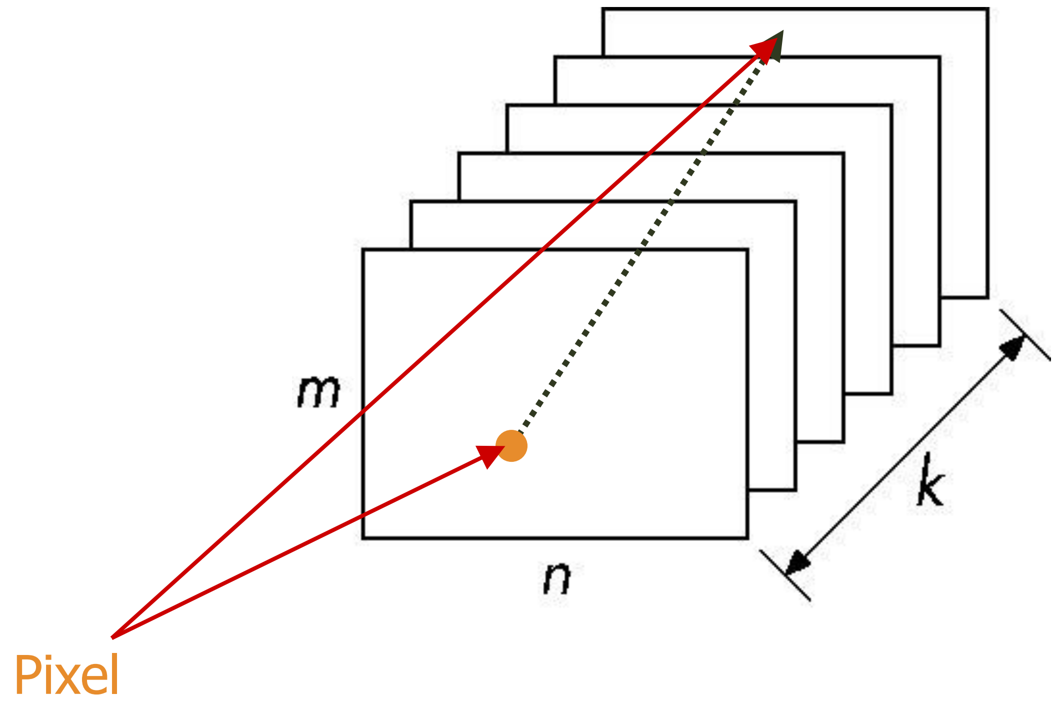
Discrete Techniques

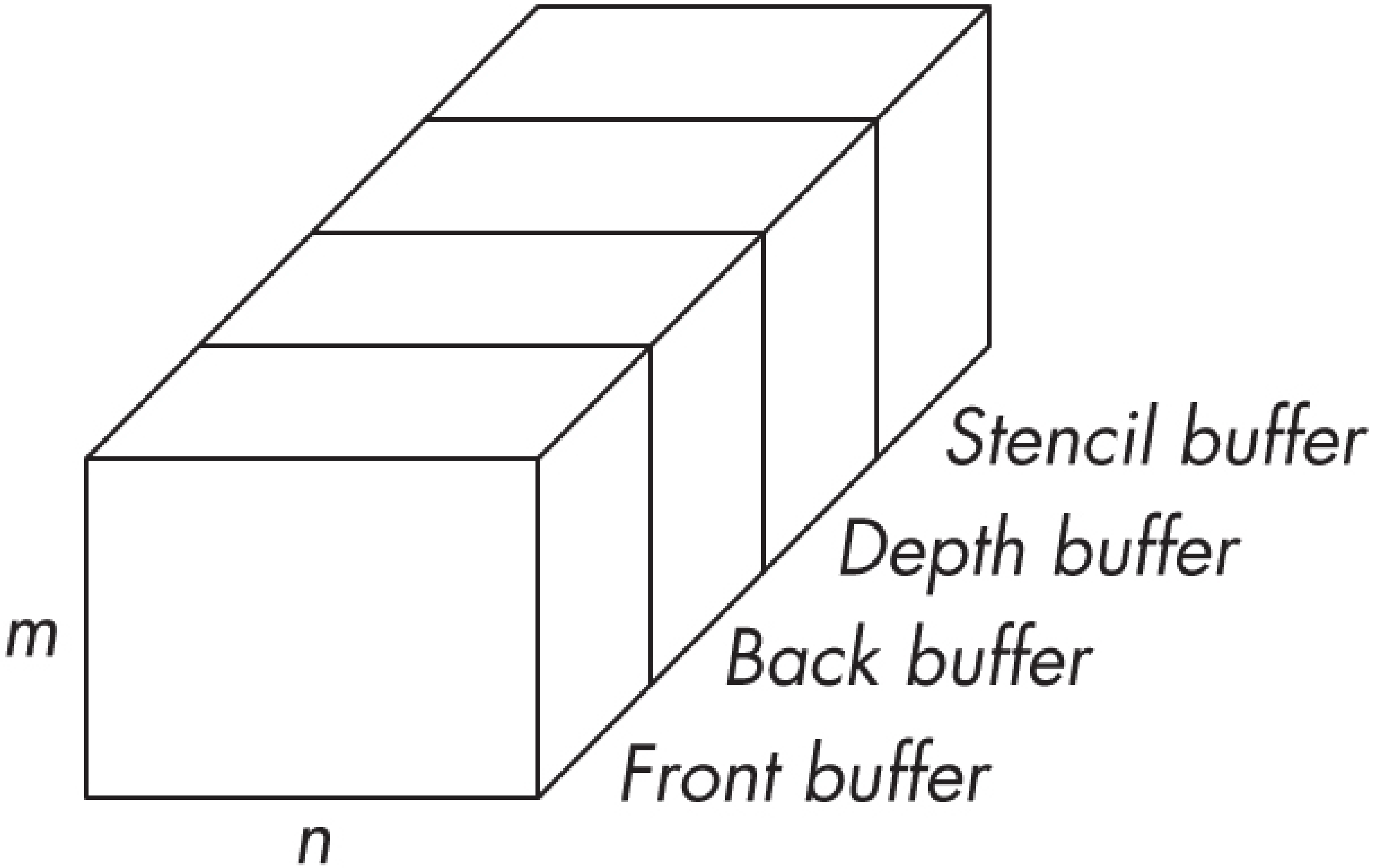
12TH WEEK, 2022

Buffer

- Define a buffer by its spatial ____________ ($n \times m$) and its __________ (or __________) k, the number of bits/pixel



WebGL Frame Buffer



Where are the Buffers?

- HTML5 Canvas
  - Default front and back color buffers
  - Under control of local window system
  - Physically on graphics card
- Depth buffer also on graphics card
- Stencil buffer
  - Holds masks
- Most RGBA buffers 8 bits per component
- Latest are floating point (IEEE)

Other Buffers

- Desktop OpenGL supported other buffers
  - Auxiliary color buffers
  - Accumulation buffer
  - These were on application side
  - Now deprecated
- GPUs have their own or attached memory
  - Texture buffers
  - Off-screen buffers
    - Not under control of window system
    - May be floating point

Images

- Framebuffer contents are unformatted
  - Usually RGB or RGBA
  - One byte per component
  - No compression
- Standard Web image format
  - Jpeg, gif, png
- WebGL has no conversion functions
  - Understands standard Web formats for texture images

Buffer Reading

- WebGL can read pixels from the framebuffer with gl.readPixels( )
- Returns only 8 bit RGBA values
- In general, the format of pixels in the frame buffer is different from that of processor memory and these two types of memory reside in different places
  - Need packing and unpacking
  - Reading can be slow
- Drawing through texture functions and off-screen memory (frame buffer object)

WebGL Pixel Function

```
gl.readPixels(x,y,width,height,format,type,myimage)
```

start pixel in frame buffer

size

type of image

type of pixels

pointer to processor memory

```
var myimage[512*512*4];

gl.readPixels(0,0, 512, 512, gl.RGBA,
      gl.UNSIGNED_BYTE, myimage);
```

Render to Texture

- GPUs now include a large amount of texture memory that we can write into
- Advantage: fast (not under control of window system)
- Using frame buffer objects (FBOs) we can render into texture memory instead of the frame buffer and then read from this memory
  - Image processing
  - GPGPU

The Limits of Geometric Modeling

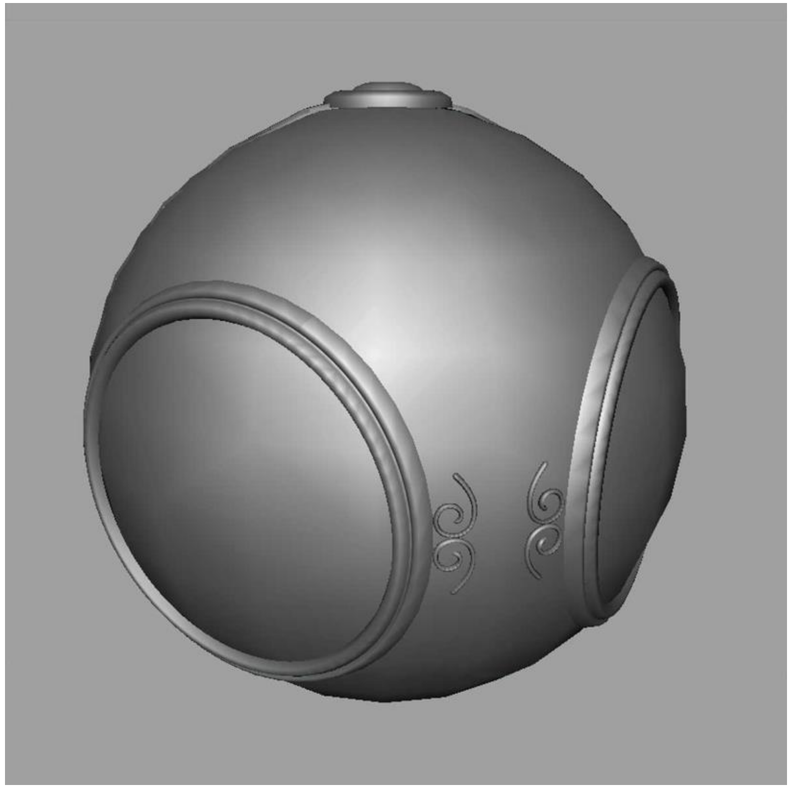
- Although graphics card can render over 10 million polygons per second, that number is insufficient for many phenomena
  - Clouds, grass, terrain, skin, etc.

- Consider the problem of modeling an orange
  - An orange-colored sphere → too simple
    ➔ *texture mapping*
  - More complex shape → too many polygons to model all the dimples
    ➔ *bump mapping*

Three Types of Mapping

- *Texture mapping*
  - Uses images to fill inside of polygons
- *Environment* (reflection) *mapping*
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
- *Bump mapping*
  - Emulates altering normal vectors during the rendering process

Texture Mapping

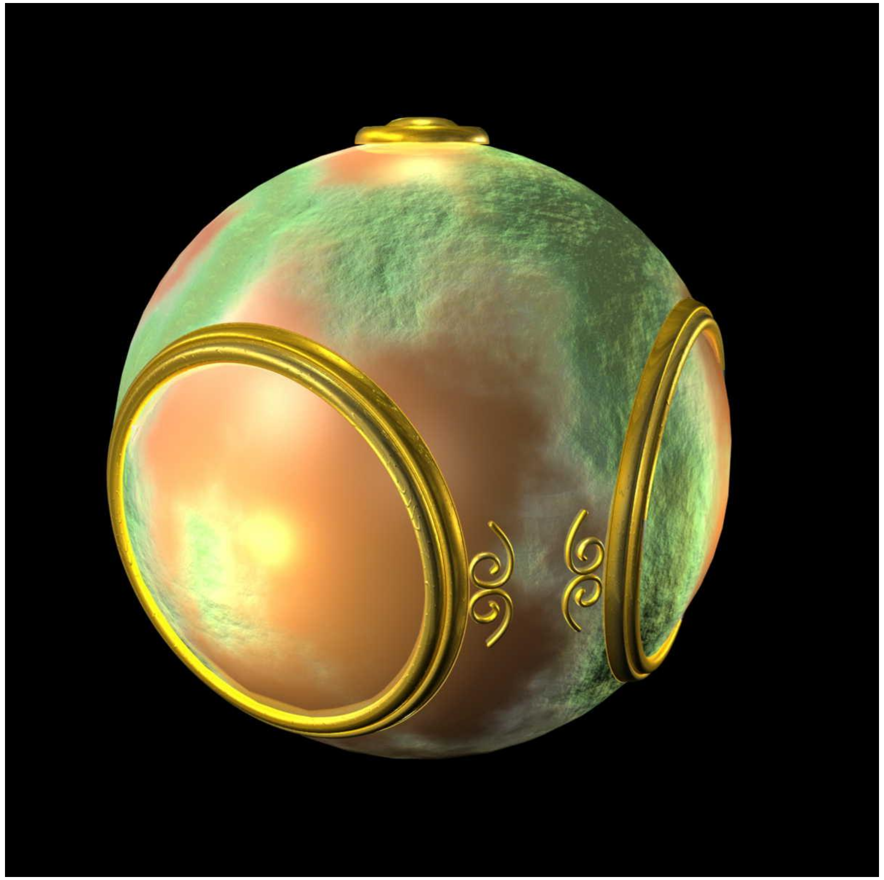


Geometric Model

Texture Mapped

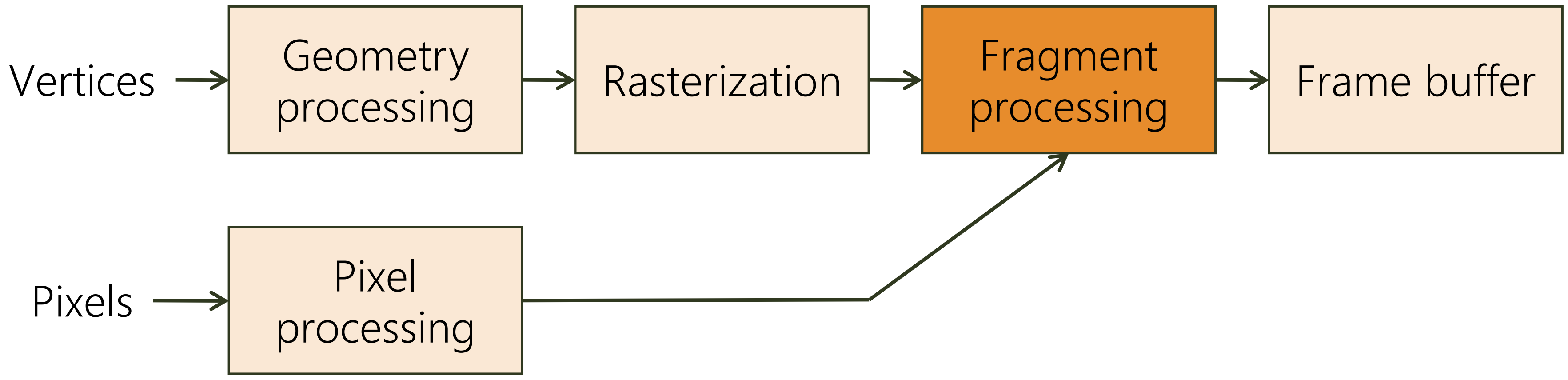
Environment Mapping

Bump Mapping



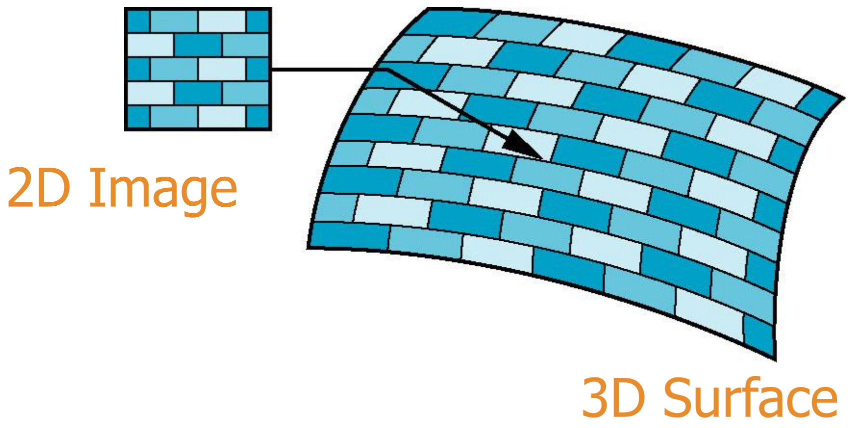
Where Does Mapping Take Place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because a few polygons make it past clipper



Is It Simple?

- Mapping a pattern (texture) to a surface

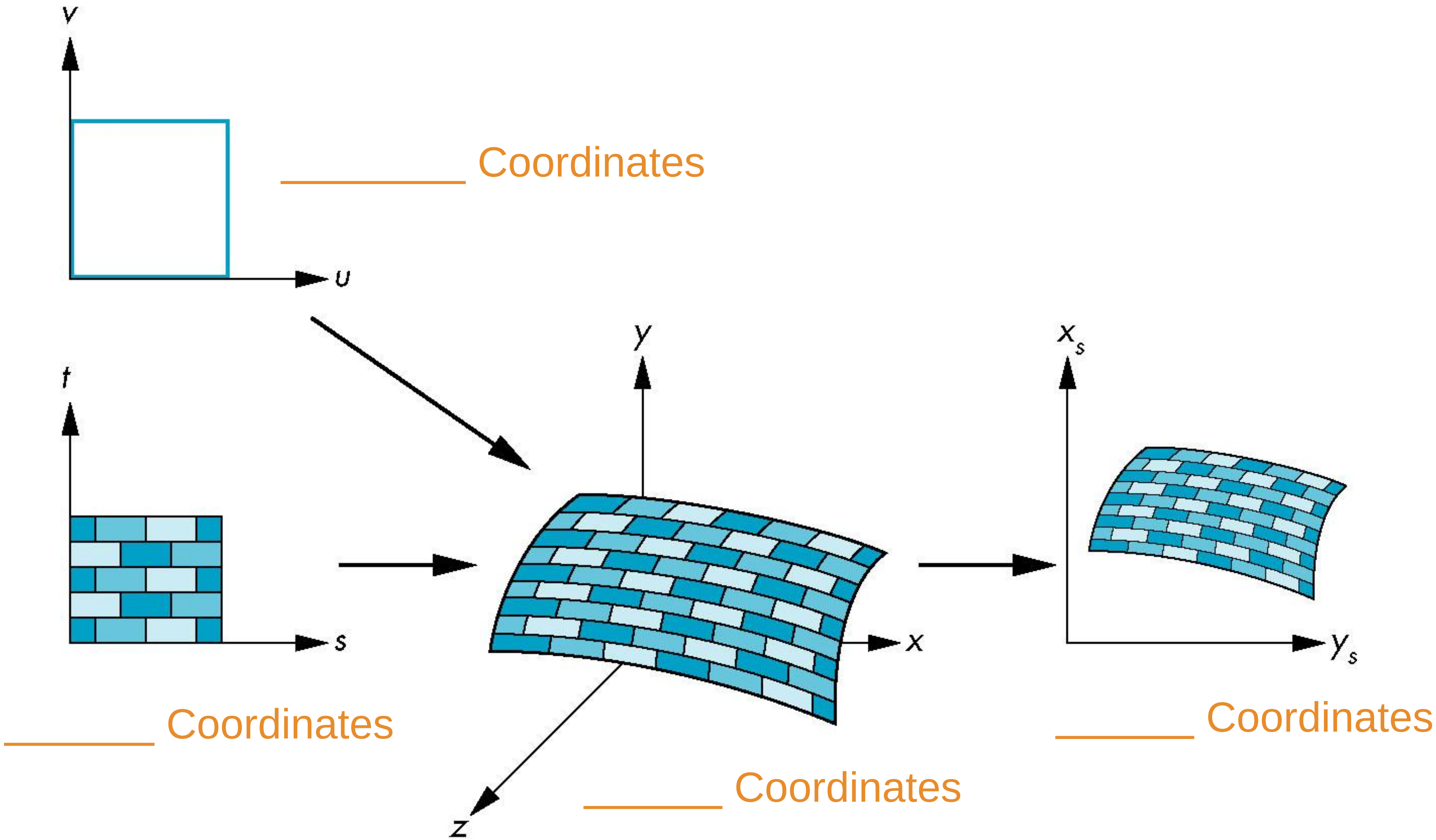


- Although the idea is simple – map an image to a surface – there are 3 or 4 coordinate system involved

Coordinate Systems

- ___________ coordinates
  - May be used to model curves and surfaces
- __________ coordinates
  - Used to identify points in the image to be mapped
- Object or _________ coordinates
  - Conceptually, where the mapping takes place
- Window or _________ coordinates
  - Where the final image is really produced

Texture Mapping



Terminology for Texture Mapping

- ______(texture element)
  - Textures are brought into processor memory as arrays
- _________ *coordinates* $T(s, t)$
  - Continuous rectangular 2D texture pattern
  - Generally varying over the interval (0, 1)
- Texture ____
  - World coordinates ↔ texture coordinates

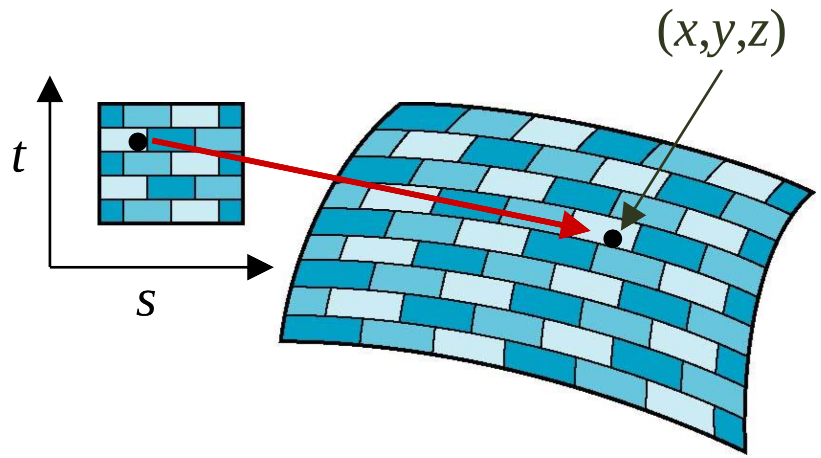
$$x = x(s,t)$$
$$y = y(s,t)$$
$$z = z(s,t)$$

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

Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point on 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

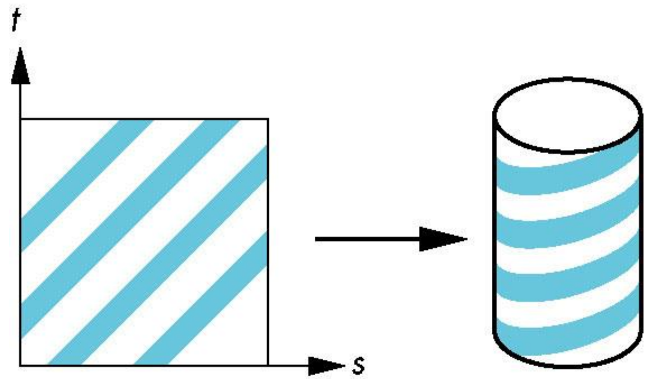
- We really want to go __________
  - Given a texel, we want to know to which point on an object it corresponds → __________
  - Given a point on an object, we want to know to which point in the texture it corresponds → __________
- Need a map of the form

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

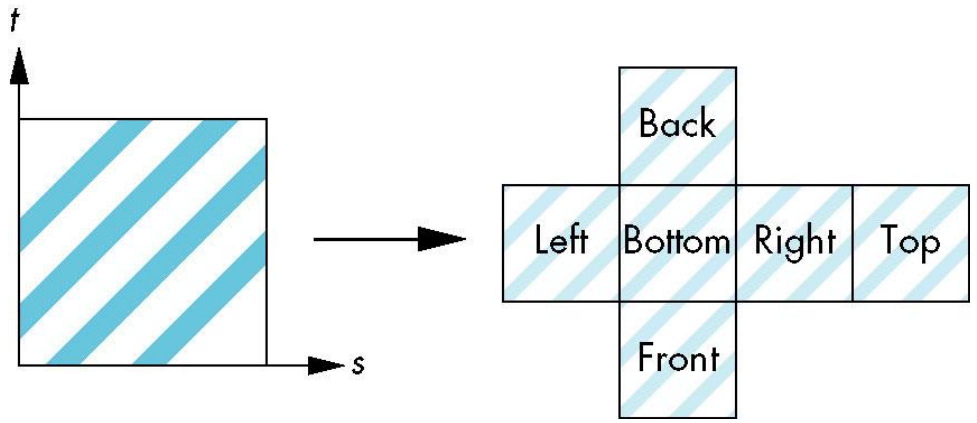
- Such functions are difficult to find in general

Two-Part Mapping

- One solution to the mapping problem is to first map the texture to a simple intermediate surface such as a cylinder, a sphere, a box
- Example:



Texture Mapping with Cylinder



Texture Mapping with a Box

First Mapping

- Cylindrical mapping
  - Parametric cylinder:

$$
\boxed{\begin{aligned} x &= r\cos 2\pi u \\ y &= r\sin 2\pi u \\ z &= v/h \end{aligned}}
$$

$s=u$
$t=v$

r: radius
h: height

- Spherical mapping
  - Parametric sphere:

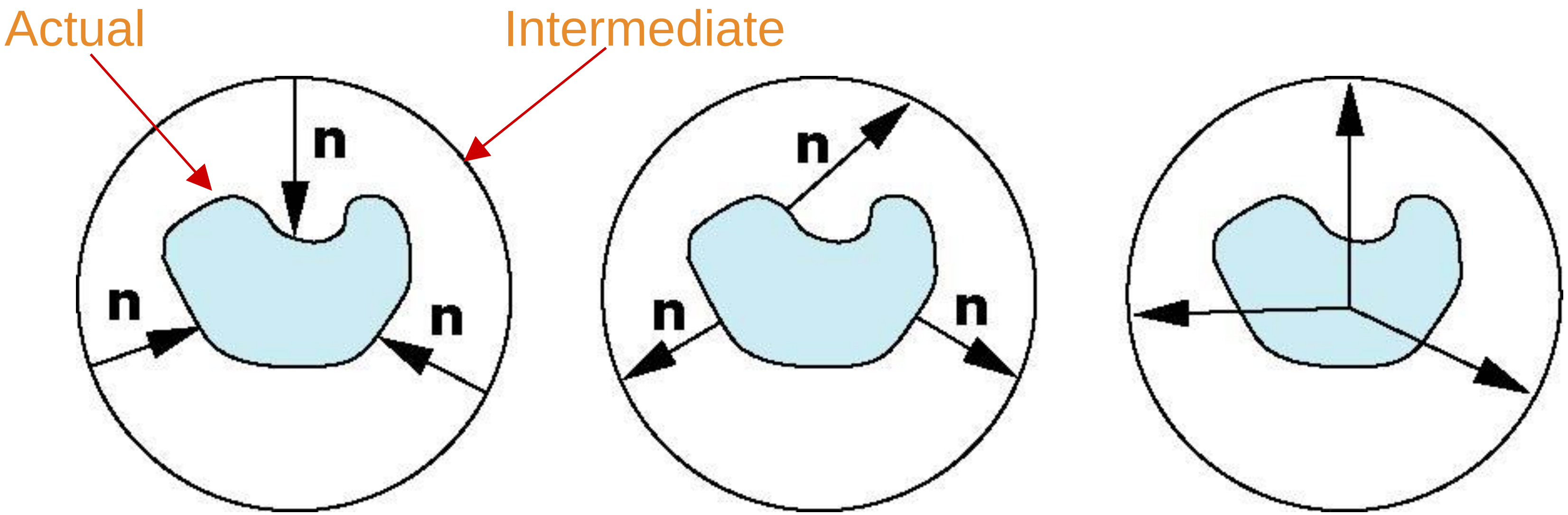
$$
\boxed{\begin{aligned} x &= r\cos 2\pi u \\ y &= r\sin 2\pi u \cos 2\pi v \\ z &= r\sin 2\pi u \sin 2\pi v \end{aligned}}
$$

$s=u$
$t=v$

  - Spheres are used in environmental maps
- Box mapping
  - Easy to use with simple orthographic projection
  - Also used in environment maps

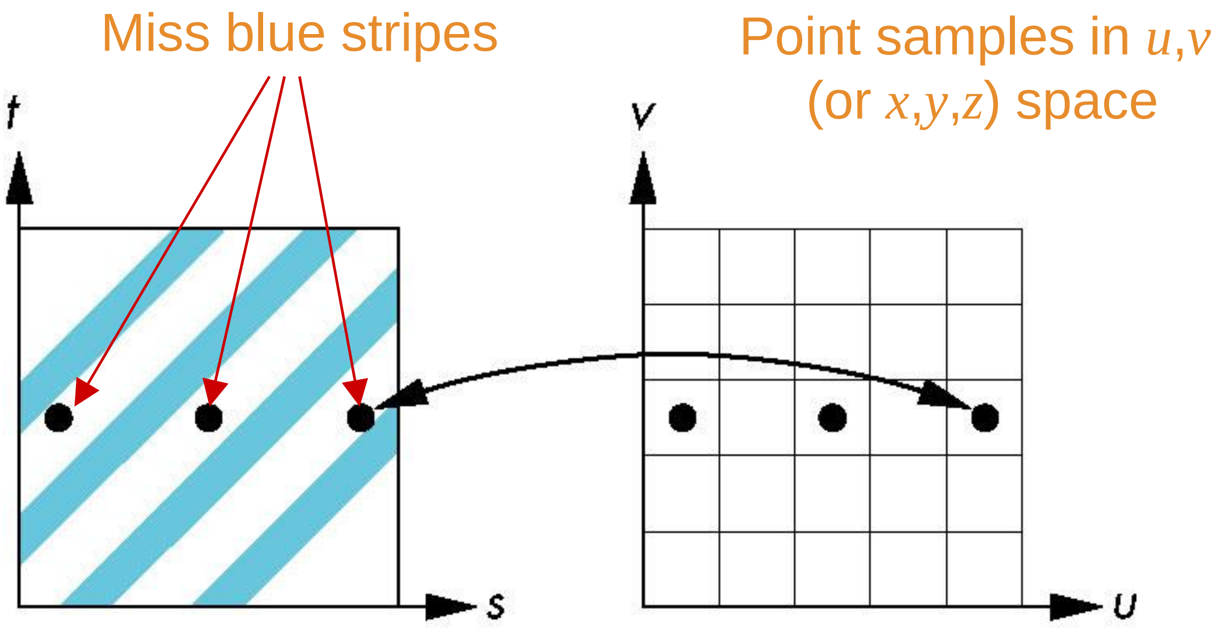
Second Mapping

- Map from intermediate object to actual object
  - Using the normals from intermediate to actual
  - Using the normals from actual to intermediate
  - Using the vectors from center of the object to intermediate



Aliasing

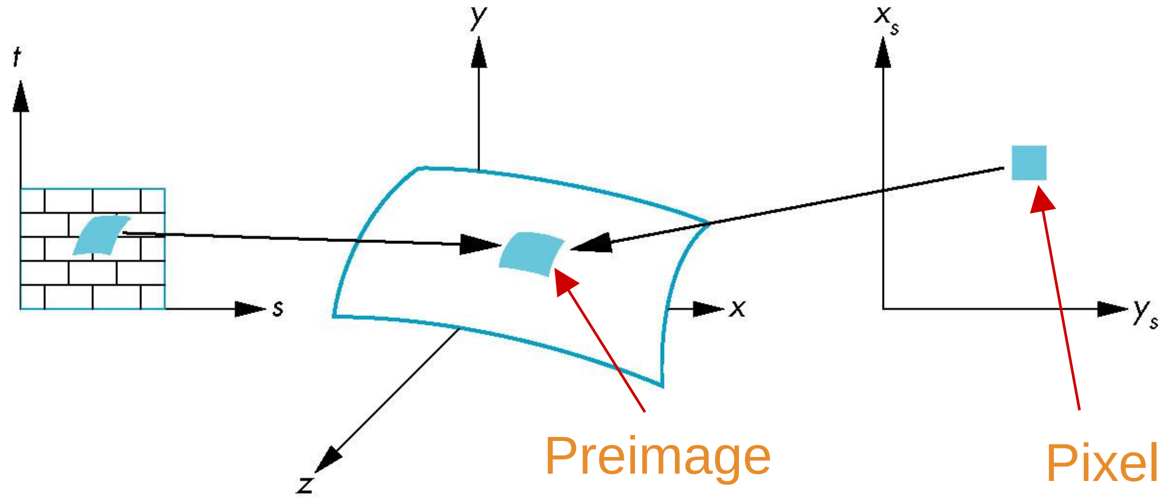
- ______ sampling of the texture can lead to aliasing errors



Point samples in texture space

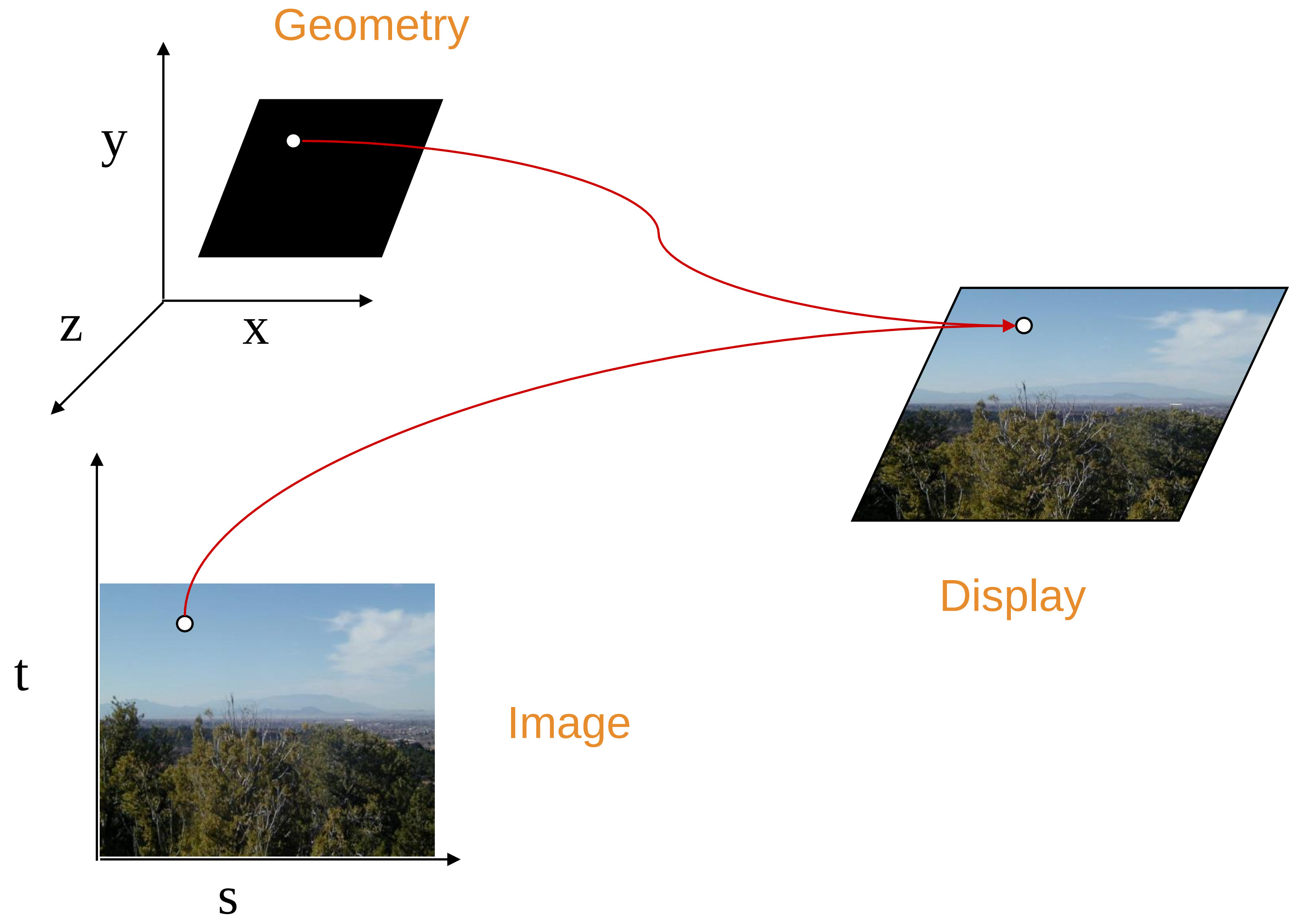
Area Averaging

- A better but slower option is to use area averaging



- *Preimage*
  - The projection of the corners of a pixel backward into object space
  - Preimage of the pixel is curved

Texture Mapping

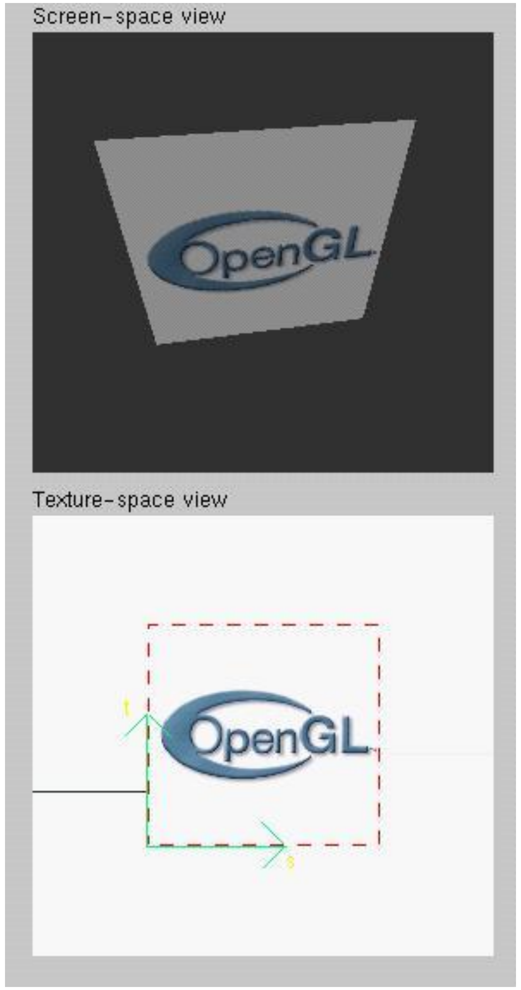


Basic Strategy

- Three steps to apply 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

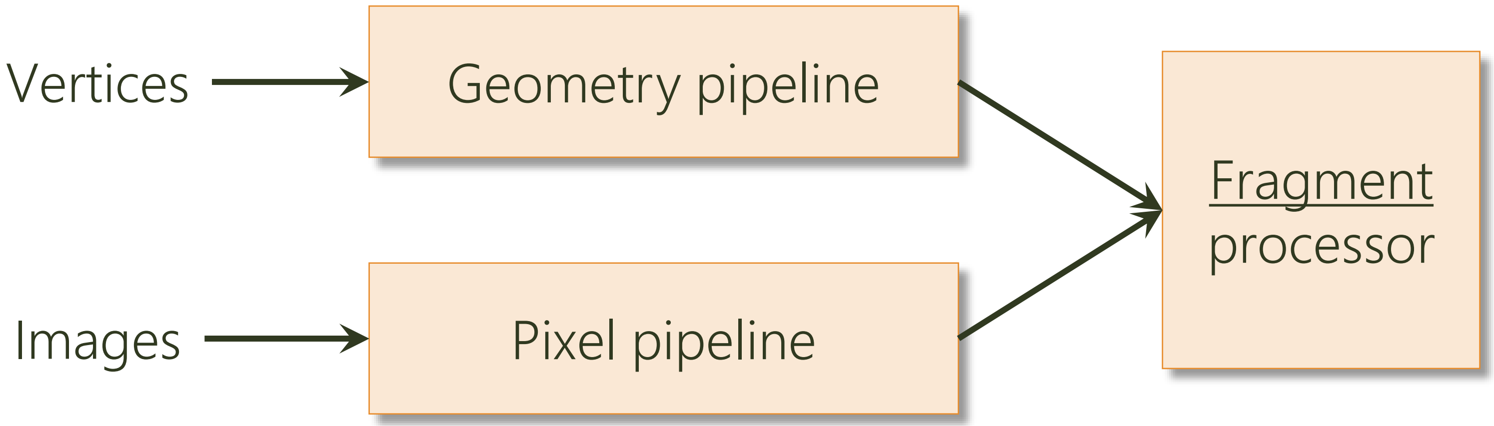
Texture Example

- The texture (below) is a 256×256 image that has been mapped to a rectangular polygon which is viewed in perspective



Texture Mapping and the WebGL Pipeline

- Images and geometry flow through separate pipelines that join during fragment processing
  - "Complex" textures do not affect geometric complexity



Specifying a Texture Image

- Define a texture image from an array of ______*s* (texture element) in CPU memory
- Use an image in a standard format such as BMP
  - Scanned image
  - Generated 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

```
void gl.texImage2D(target, level, components,
                   width, height, 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

ex)
```
gl.texImage2D(gl.TEXTURE_2D, 0, 3, 512, 512, 0, gl.RGB,
              gl.UNSIGNED_BYTE, my_texels);
```

A Checkboard Image

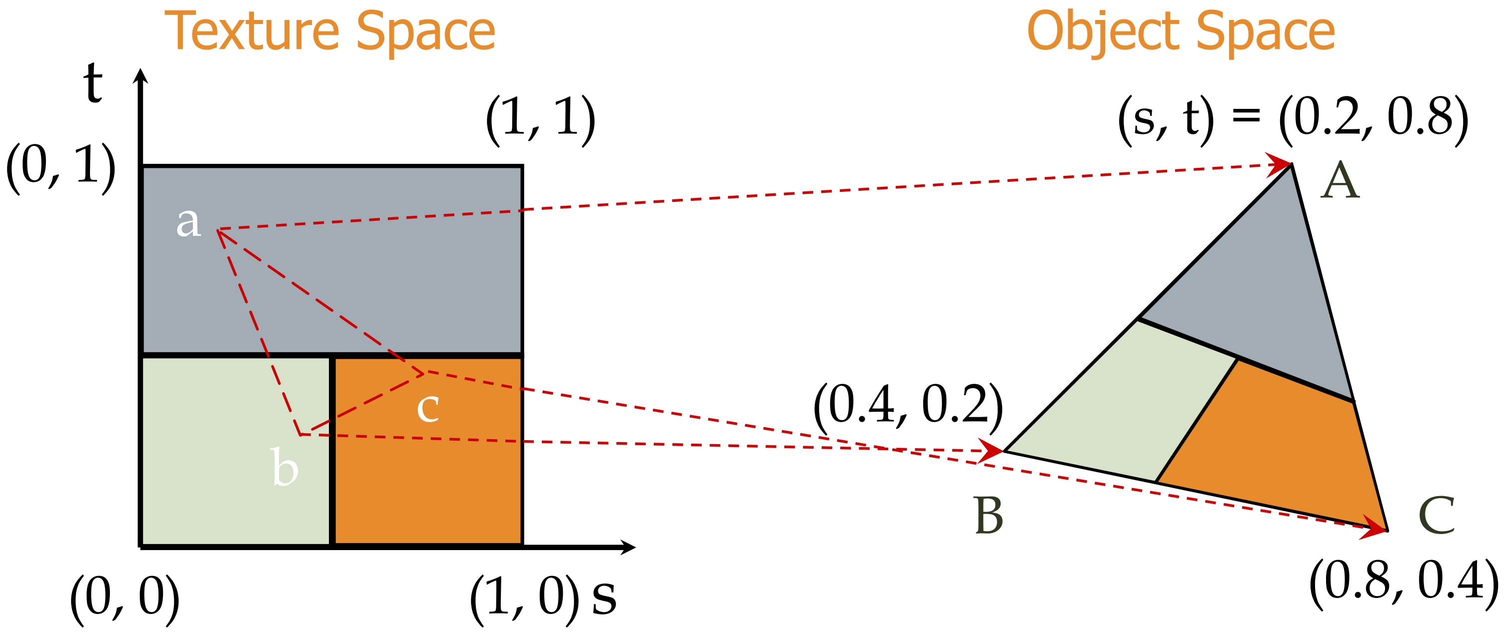
```
var image1 = new Uint8Array(4*texSize*texSize);
    for ( var i = 0; i < texSize; i++ ) {
        for ( var j = 0; j <texSize; j++ ) {
            var patchx = Math.floor(i/(texSize/numChecks));
            var patchy = Math.floor(j/(texSize/numChecks));
            if(patchx%2 ^ patchy%2) c = 255;
            else c = 0;
            //c = 255*(((i & 0x8) == 0) ^ ((j & 0x8)  == 0))
            image1[4*i*texSize+4*j] = c;
            image1[4*i*texSize+4*j+1] = c;
            image1[4*i*texSize+4*j+2] = c;
            image1[4*i*texSize+4*j+3] = 255;
        }
    }
```

Using a GIF Image

```
// specify image in JS file

var image = new Image();
    image.onload = function() {
         configureTexture( image );
    }
    image.src = "SA2011_black.gif”

// or specify image in HTML file with <img> tag

// <img id = "texImage" src = "SA2011_black.gif"></img>

var image = document.getElementById("texImage”)
window.onload = configureTexture( image );
```

Mapping a Texture

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

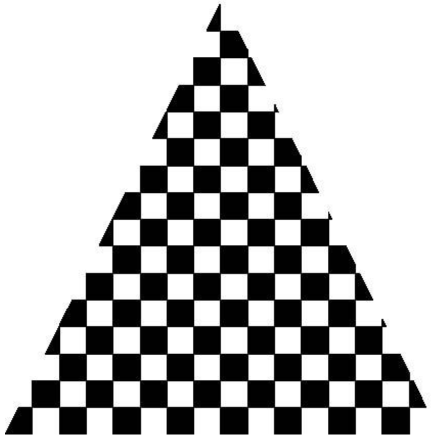


Cube Example

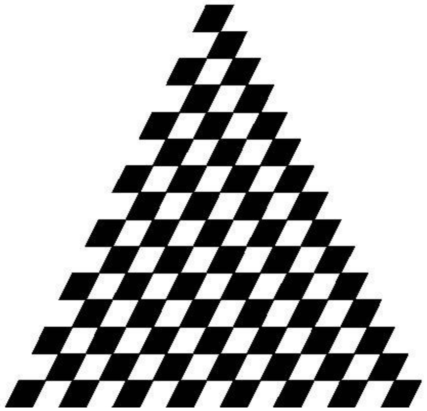
```
var texCoord = [
    vec2(0, 0),
    vec2(0, 1),
    vec2(1, 1),
    vec2(1, 0)
];

function quad(a, b, c, d) {
     pointsArray.push(vertices[a]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[0]);

     pointsArray.push(vertices[b]);
     colorsArray.push(vertexColors[a]);
     texCoordsArray.push(texCoord[1]);
// etc
```

Interpolation

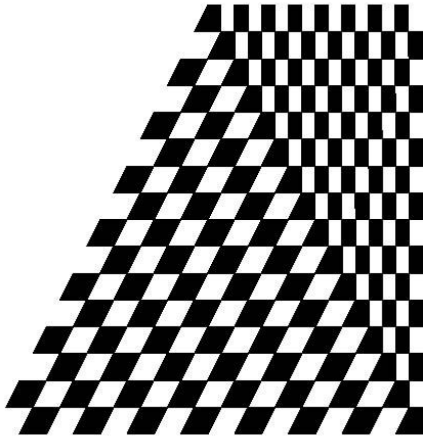
- WebGL uses ____________ to find proper texels from specified texture coordinates
  - Can be distortion



Good selection of tex coordinates



Poor selection of tex coordinates



Texture stretched over trapezoid showing effects of bilinear interpolation

Using Texture Objects

1. Specify ______s in texture objects
2. Set texture ______
3. Set texture function
4. Set texture _____ mode
5. Set optional perspective correction hint
6. ______ texture object
7. _______ texturing
8. Supply texture _________s for vertex
   - Coordinates can also be generated

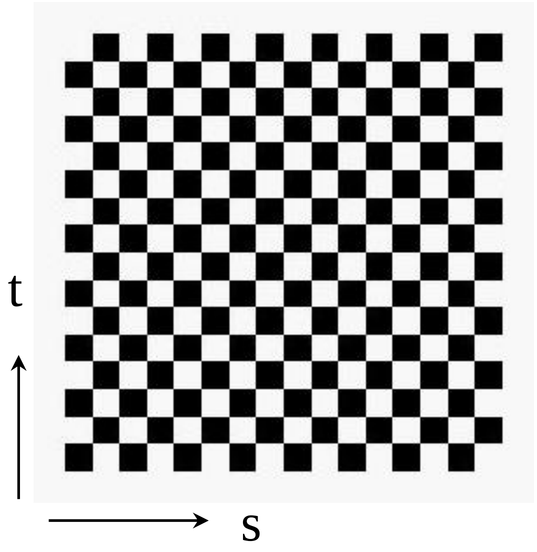
Texture Parameters

- WebGL has a variety of parameters that determine how texture is applied
  - ____________ ____________s determine what happens if s and t are outside the (0, 1) range
  - ________ ________s allow us to use area averaging instead of point samples
  - _____________ allows us to use textures at multiple resolutions
  - ______________ parameters determine how texture mapping interacts with shading

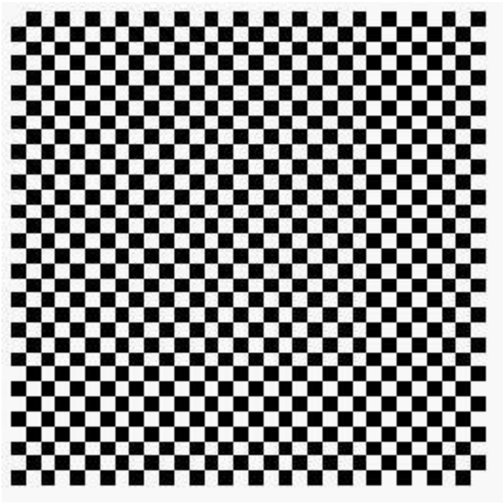
Wrapping Modes

- __________: if $s, t > 1$ use 1, if $s, t < 0$ use 0
- Wrapping: use s, t modulo 1

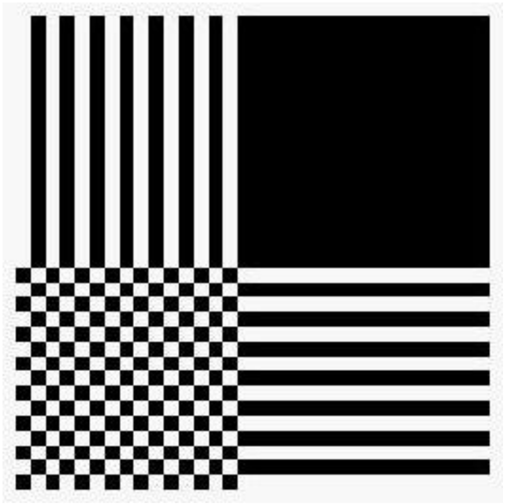
```
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT );
```



texture



gl.REPEAT wrapping



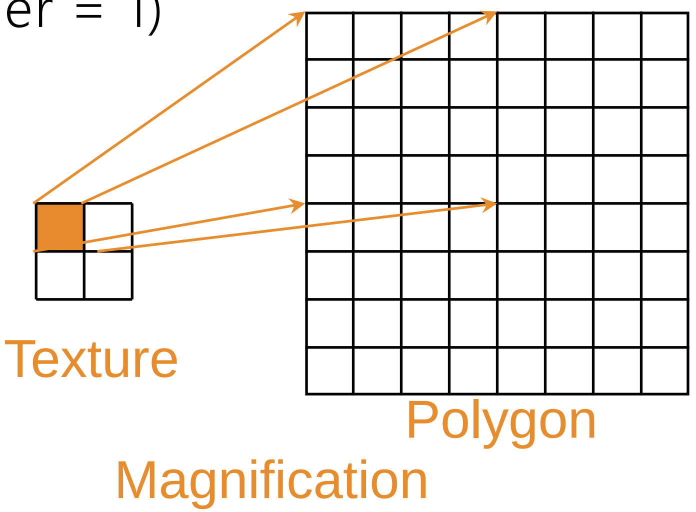
gl.CLAMP wrapping

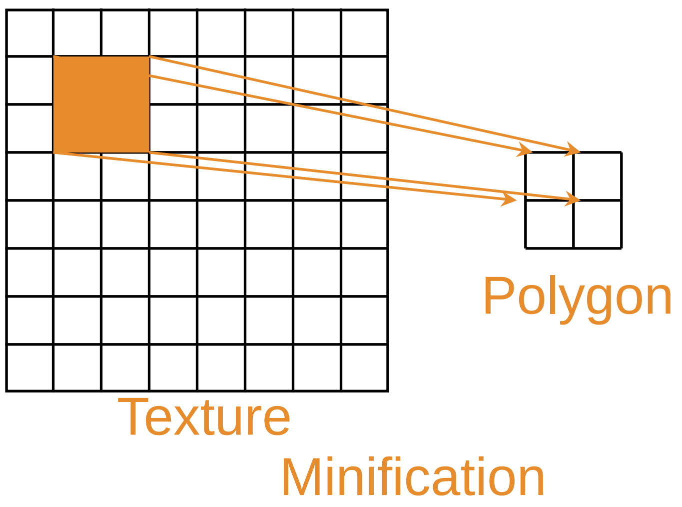
Filter Modes

- Mode determined by `gl.texParameteri( target, type, mode )`

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MIN_FILTER, gl.LINEAR);
```

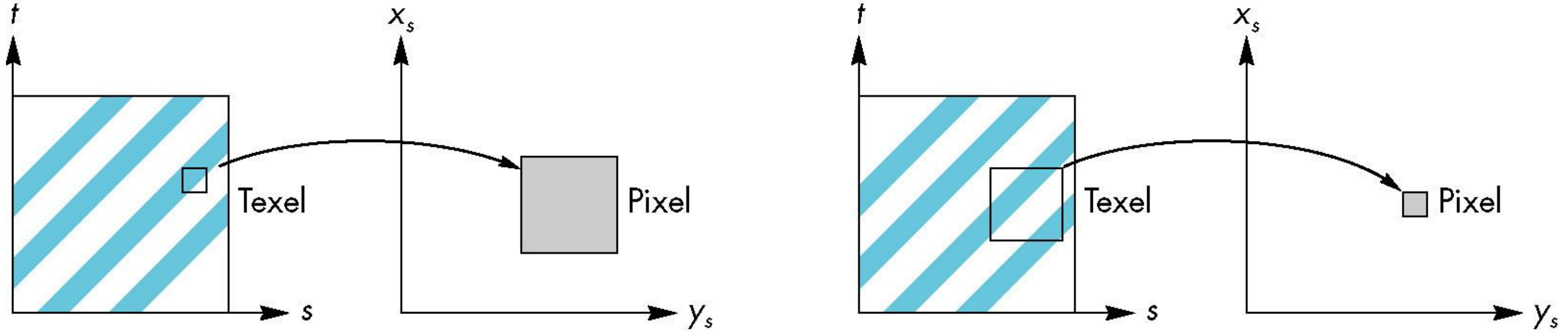
- Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1)



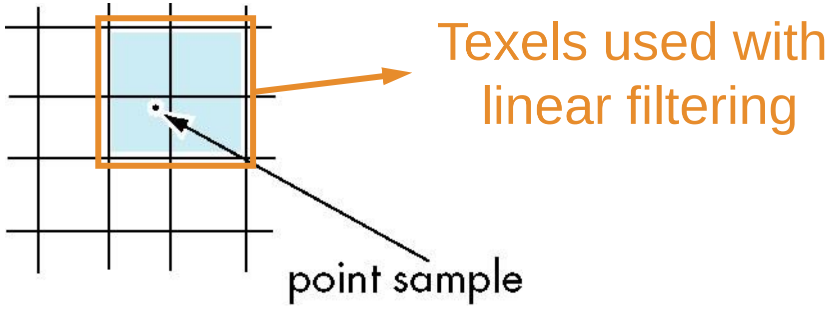


Magnification and Minification

- More than one texel can cover a pixel (__________) or more than one pixel can cover a texel (___________)



- Can use _____ _________ (nearest texel) or _____ ________ (2x2 filter) to obtain texture values



Mipmapped Textures

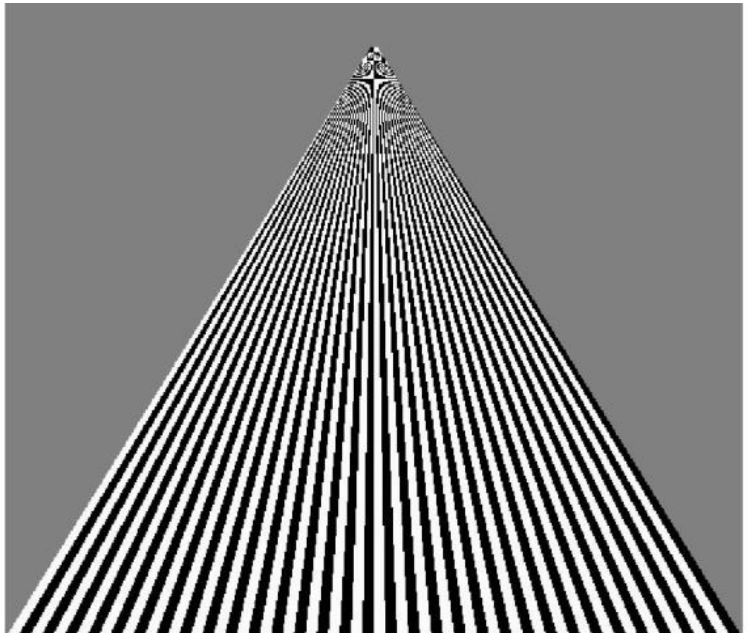
- ___________ allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

```
gl.texImage2D(gl.TEXTURE_*D, level, … )
```

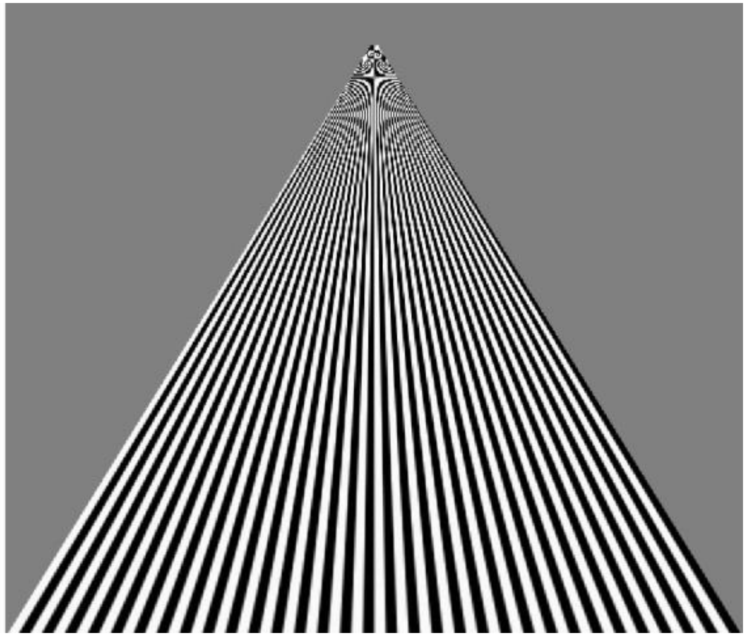
Mipmapped Textures

- Fast and easy for hardware

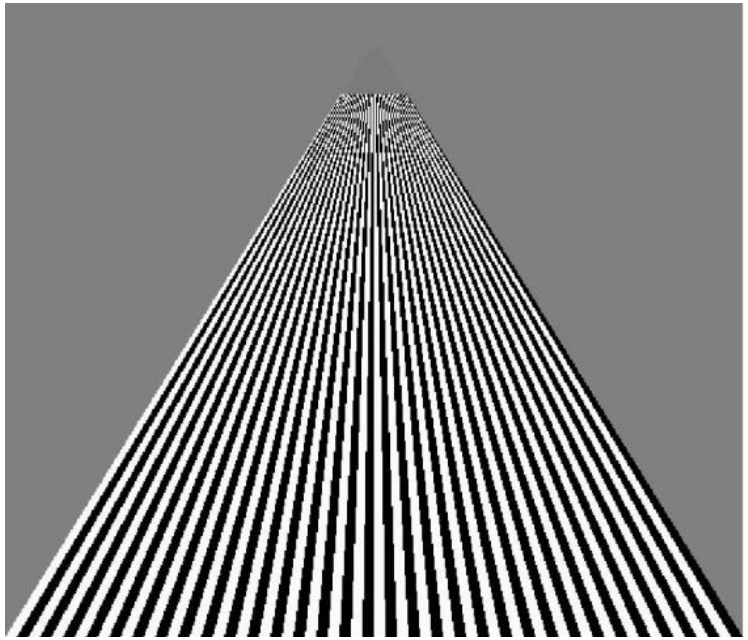
Example



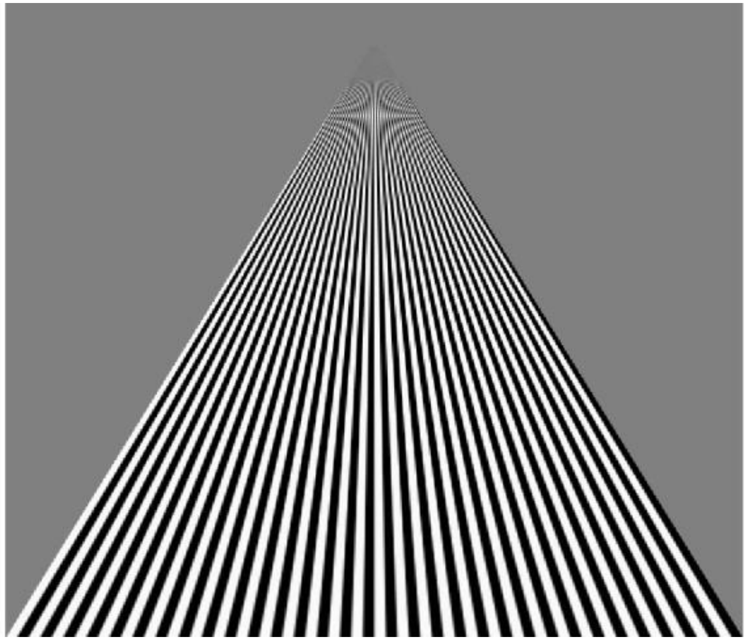
Point sampling



Linear filtering



Mipmapped point sampling



Mipmapped linear filtering

Environment Modes

- Texture can be applied many ways
    - Texture fully determines color
    - Modulated with a computed color
    - Blended with and environmental color
- Fixed function pipeline has a function glTexEnv to set mode
    - Deprecated
    - Can get all desired functionality via fragment shader
- Can also use multiple texture units

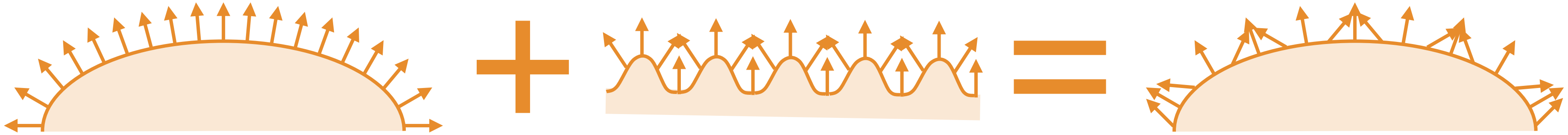
Other Texture Features

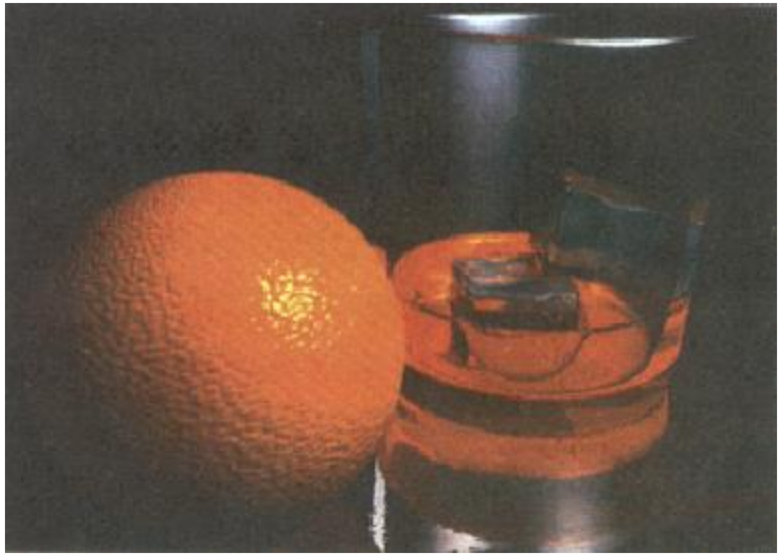
- ______________ *maps*
  - Start with image of environment through wide angle lens
    - Can be either a real scanned image or an image created in OpenGL
  - Use this texture to generate a spherical map
  - Use automatic texture coordinate generation

- ______________
  - Apply a sequence of textures through cascaded texture units

Bump Mapping

- Render objects so that they appear to have fine details (______s) that give the surface a rough appearance affected by the light position





Applying Textures

- Textures are applied during fragments shading by a ________
- Samplers return a _______ ______ from a texture object

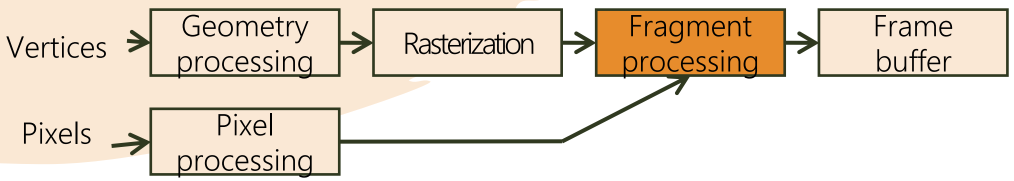
```
varying vec4 color;  //color from rasterizer
varying vec2 texCoord; //texure coordinate from rasterizer
uniform sampler2D texture; //texture object from application

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

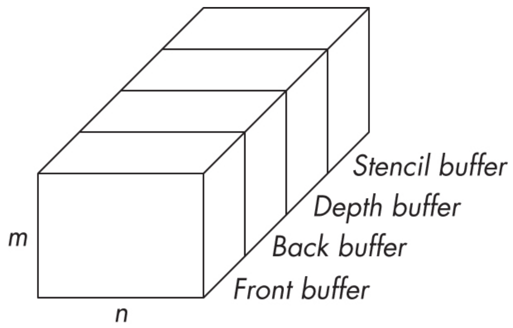
Vertex Shader

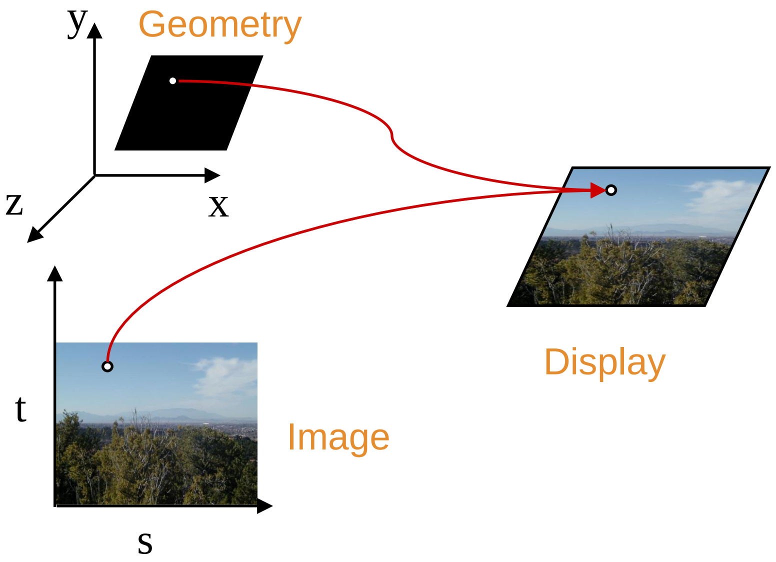
- Usually vertex shader will output texture coordinates to be ___________
- Must do all other standard tasks too
  - Compute vertex _________
  - Compute vertex _____ if needed

```
attribute vec4 vPosition; //vertex position in object coordinates
attribute vec4 vColor;    //vertex color from application
attribute vec2 vTexCoord; //texture coordinate from application

varying vec4 color; //output color to be interpolated
varying vec2 texCoord; //output tex coordinate to be interpolated
```

- WebGL frame buffer







- **Texture mapping**

1. Specify the texture
   - Read or generate image
   - Assign to texture
   - Enable texturing

```
void gl.texImage2D(target, level, components,
                    width, height, border,
                    format, type, texels);
```

2. Assign texture coordinates to vertices
   - Proper mapping function is left to application
3. Specify texture parameters
   - Wrapping, filtering

```
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT );
```

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MIN_FILTER, gl.LINEAR);
```