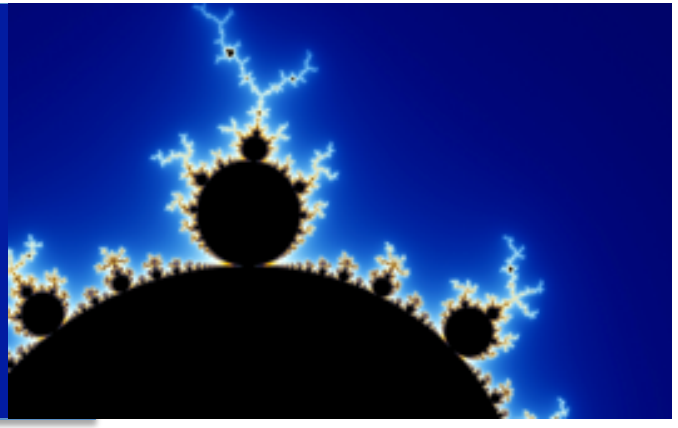
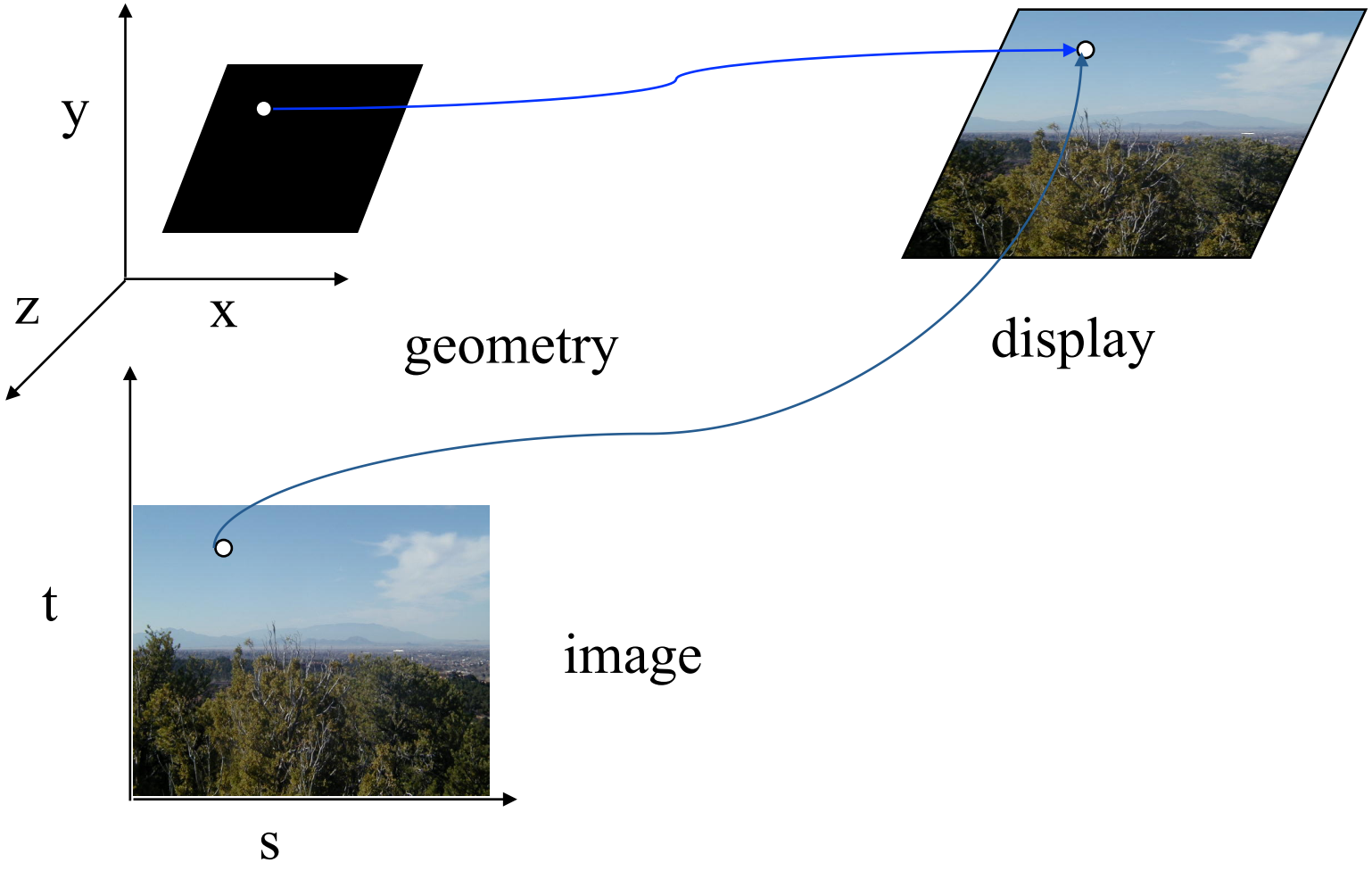
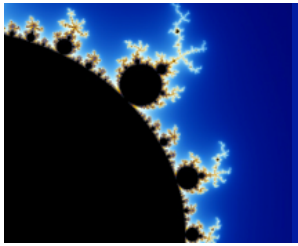
Computer Graphics



WebGL Texture Mapping

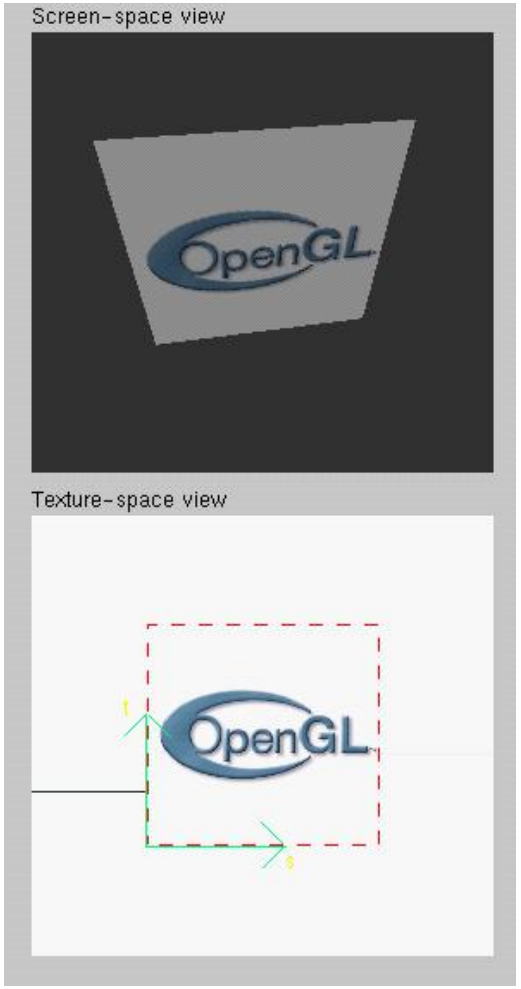
Texture Mapping



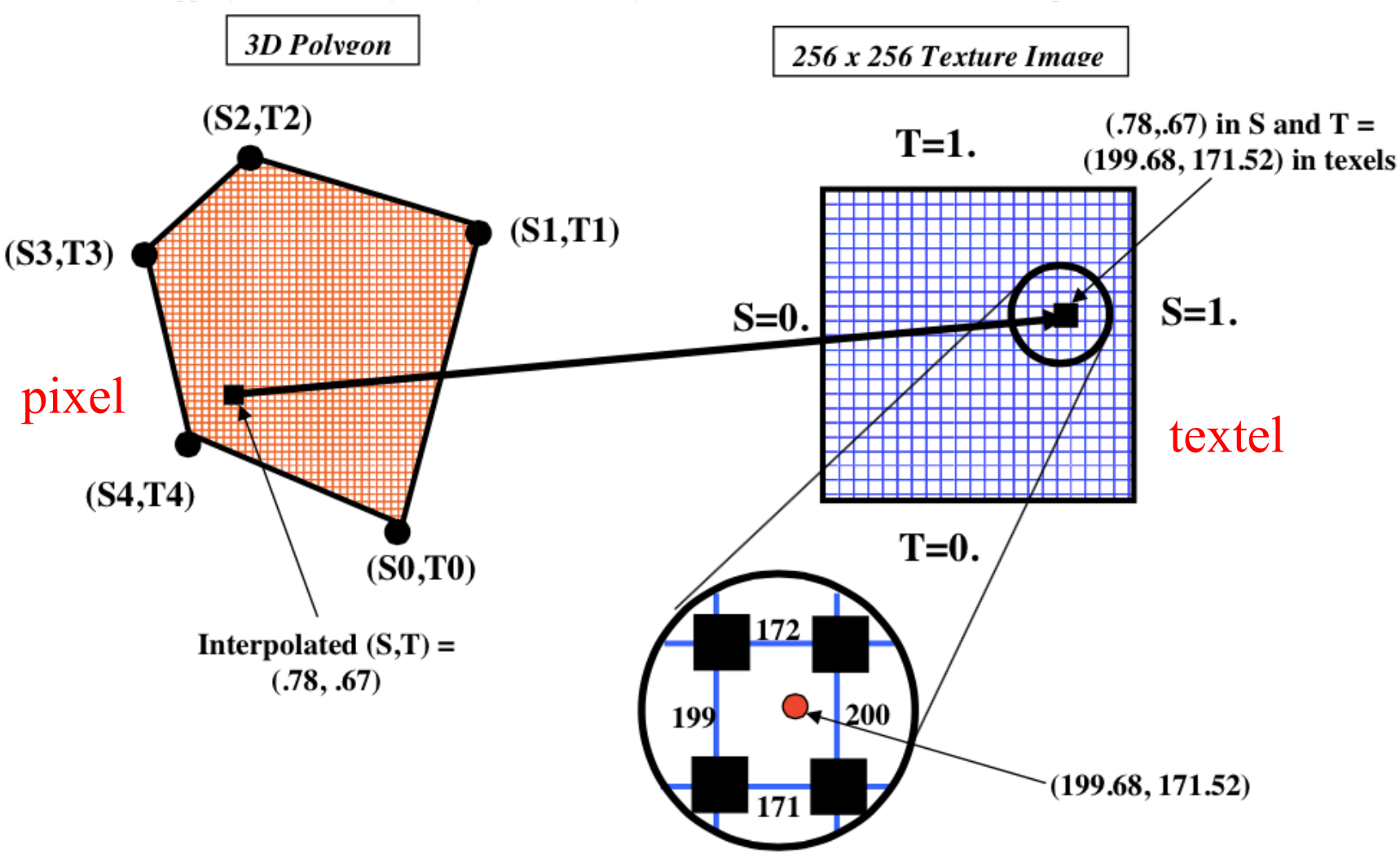


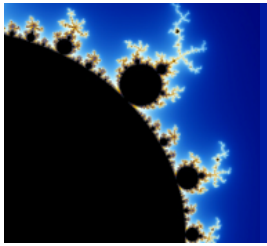
Texture Example

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



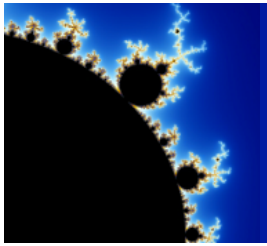
Texture Mapping





Basic steps to applying a texture

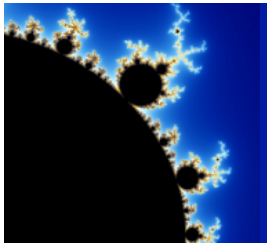
1. specify the texture
   - Create texture : self generate or use existing image
     - Define a texture image from an array of *texels* (texture elements) in CPU memory
     - Use an image in a standard format such as JPEG
     - WebGL supports only 2 dimensional texture maps
       - » no need to enable as in desktop OpenGL
       - » desktop OpenGL supports 1-4 dimensional texture maps
   - Bind to the active texture
2. assign texture coordinates to vertices
3. specify texture parameters
   - wrapping, filtering



Step 1: Using a GIF image

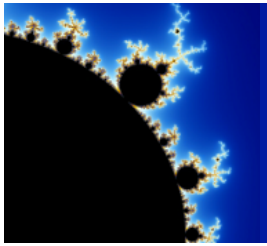
```
// specify image in JS file

var image = new Image();
image.onload = function() { configureTexture( image );  }
image.src = 'sky.gif';
```

- Image may be .jpg, .gif, .png, etc.
- Image needs to be of size that is power of 2; otherwise special filterings need to be applied



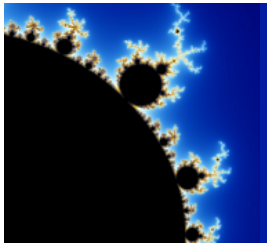
Step 1: Generating an image

```
// Create a checkerboard pattern using floats
var image1 = new Array();
    for (var i =0; i<texSize; i++)
        image1[i] = new Array();

    for (var i =0; i<texSize; i++)
        for ( var j = 0; j < texSize; j++)
            image1[i][j] = new Float32Array(4);

    // images1[i][j] = 0  or images[i][j] = 1
    for (var i =0; i<texSize; i++)
      for (var j=0; j<texSize; j++) {
        var c = (((i & 0x8) == 0) ^ ((j & 0x8)  == 0));
        image1[i][j] = [c, c, c, 1];
    }
```



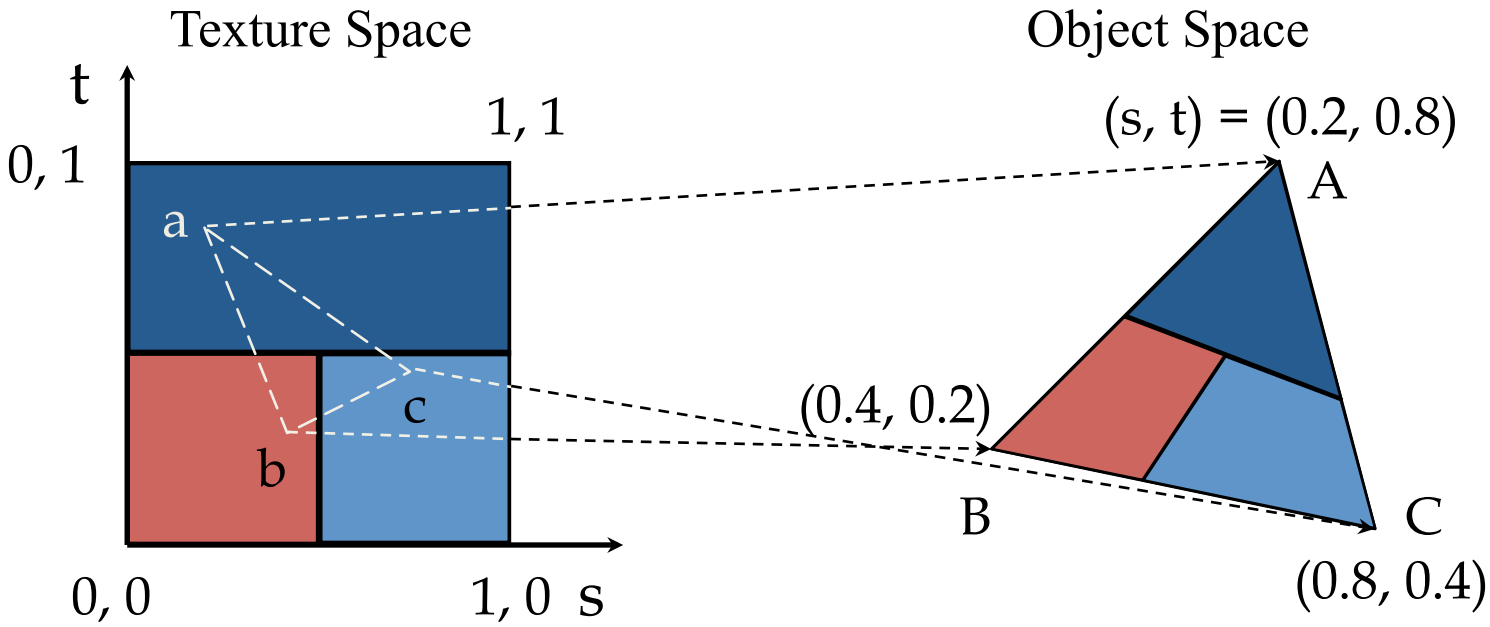
Step 1: Generating an image (cont.)

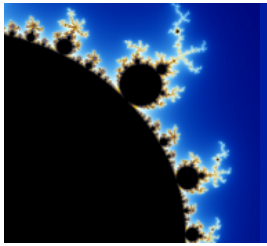
```
// Convert floats to ubytes for texture
var image2 = new Uint8Array(4*texSize*texSize);

    // images1[i][j]=0  or images[i][j]=255
    for ( var i = 0; i < texSize; i++ )
        for ( var j = 0; j < texSize; j++ )
            for(var k =0; k<4; k++)
                image2[4*texSize*i+4*j+k] = 255*image1[i][j][k];
```



Step 2: Mapping a Texture

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

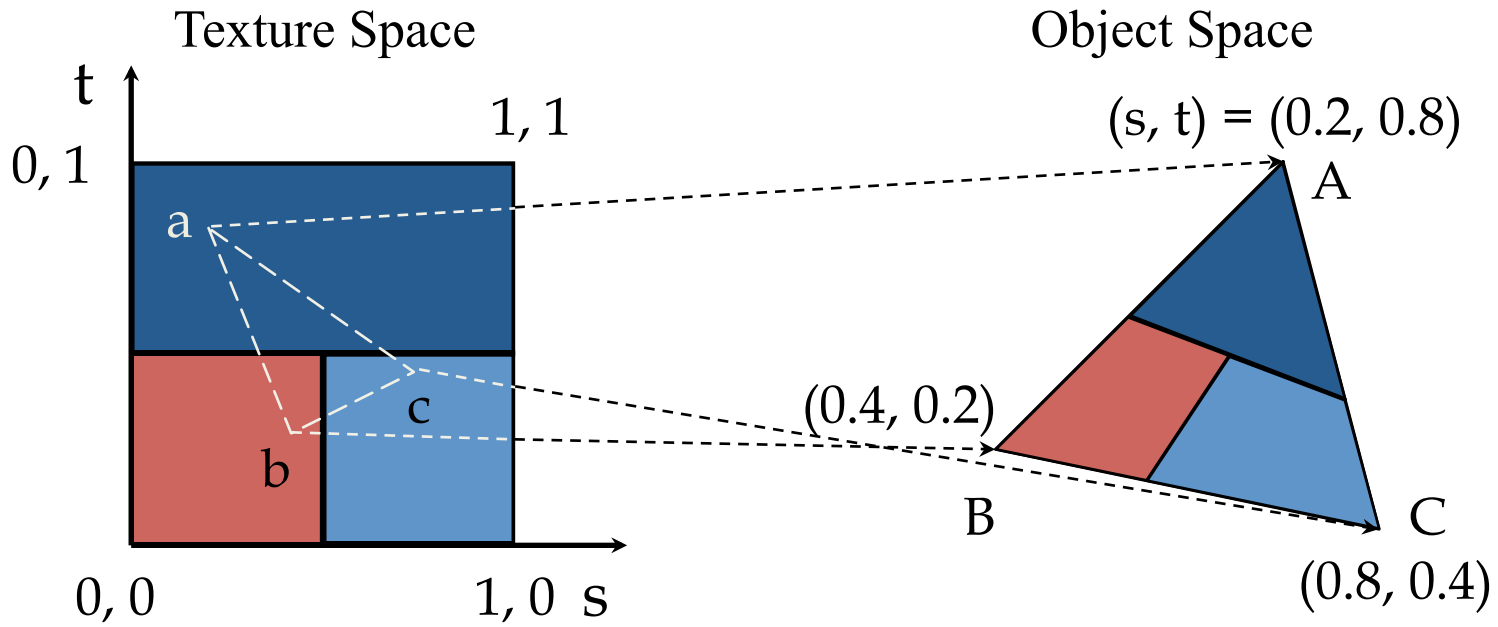


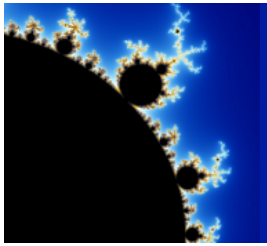


Step 2: Mapping a Texture

```
var vertices = [
    vec2(1.5, 2),
    vec2(-0.5, 0.2),
    vec2(2.4, -0.2)
];
```

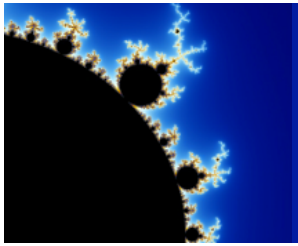
```
var texCoord = [
    vec2(0.2, 0.8),
    vec2(0.4, 0.2),
    vec2(0.8, 0.4),
];
```





Step 3: Specify texture parameters

```
function configureTexture( image ) {
    var texture = gl.createTexture();
    gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB,
                   gl.UNSIGNED_BYTE, image );

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture( gl.TEXTURE_2D, texture );

    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.generateMipmap( gl.TEXTURE_2D );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
                      gl.NEAREST_MIPMAP_LINEAR );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
                      gl.NEAREST );

    // additional filters maybe specified here
    gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
}
```



Define Image as a Texture

```
gl.texImage2D( target, level, internalformat, w, h,
   border, format, type, texels );
```

`target:` type of texture, e.g. **`GL_TEXTURE_2D`**

`level:` used for mipmapping (discussed later)

`internalformat:` Specifies the internal format of the texture. (GL_ALPHA, GL_LUMINANCE, GL_LUMINANCE_ALPHA, GL_RGB, GL_RGBA)

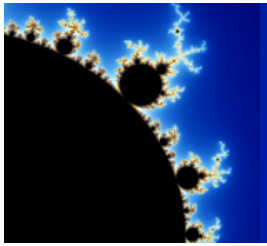
`w, h:` width and height of **`texels`** in pixels

`border:` specifies the width of the border, must be 0

`format & type:` describe the format and data type of the texel data

`texels:` pointer to the image data in memory

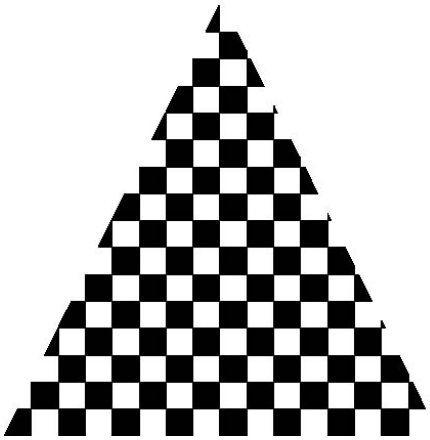
```
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0,
              gl.RGBA, gl.UNSIGNED_BYTE, image);
```



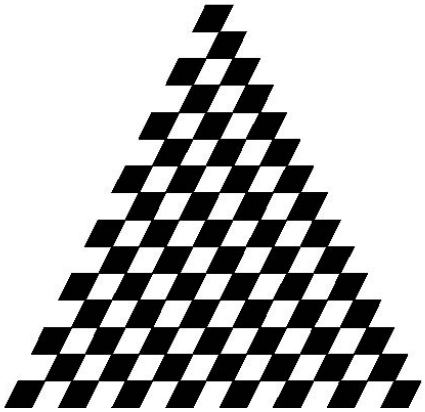
Interpolation

WebGL uses interpolation to find proper texels from specified texture coordinates ➔Can be distortions

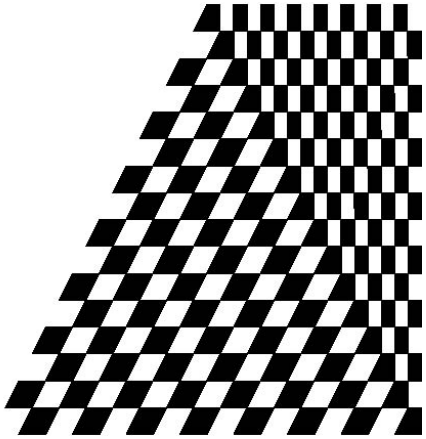
good selection of tex coordinates

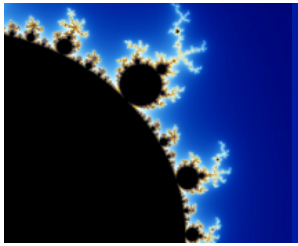


poor selection of tex coordinates



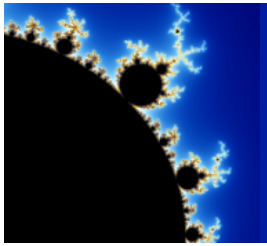
texture stretched over trapezoid





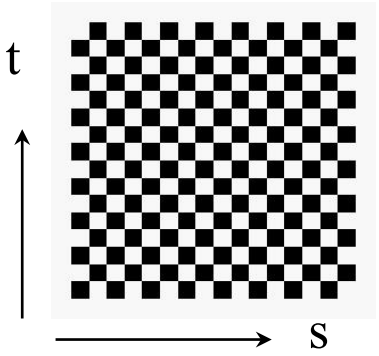
Texture Parameters

- WebGL 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

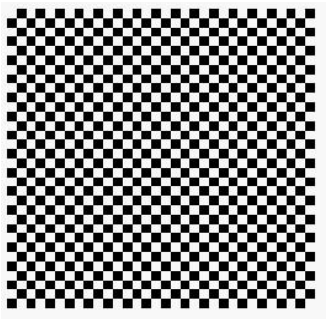


Wrapping Mode

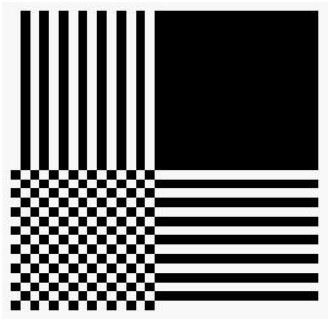
- Clamping: if s,t > 1 use 1, if s,t <0 use 0
- Wrapping: use s,t modulo 1
  - gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE)
  - gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.MIRRORED_REPEAT)



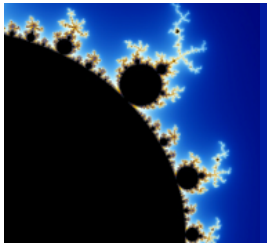
texture



gl.MIRRORED_REPEAT wrapping



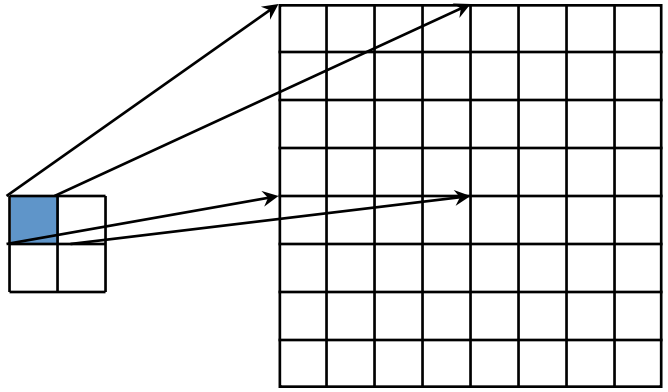
gl.CLAMP_TO_EDGE wrapping



Magnification and Minification

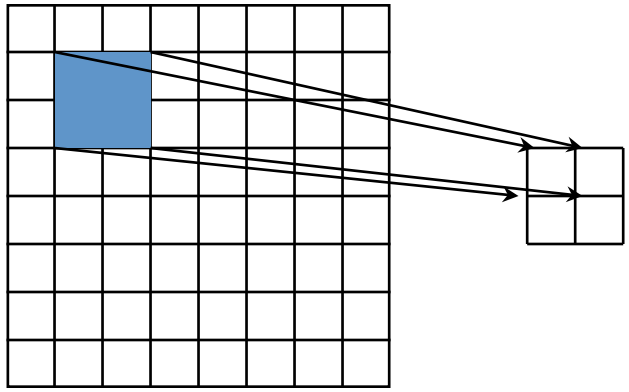
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

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



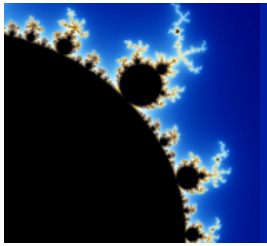
Texture Polygon

Magnification



Texture Polygon

Minification



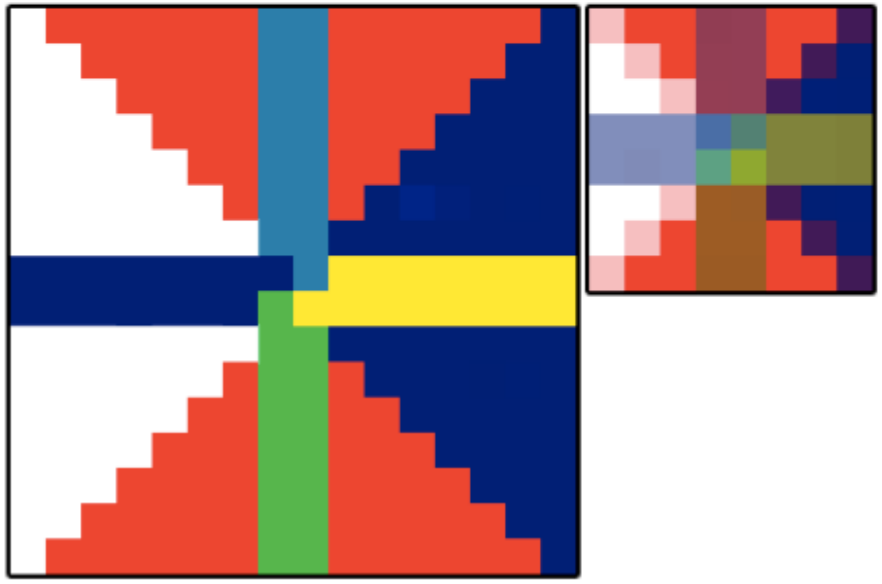
Filter Modes

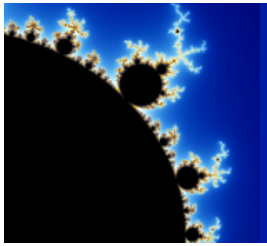
Modes determined by

```
gl.texParameteri( target, type, mode )
```

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

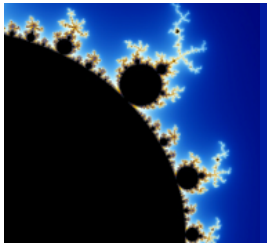
```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXURE_MIN_FILTER,
                 gl.LINEAR);
```



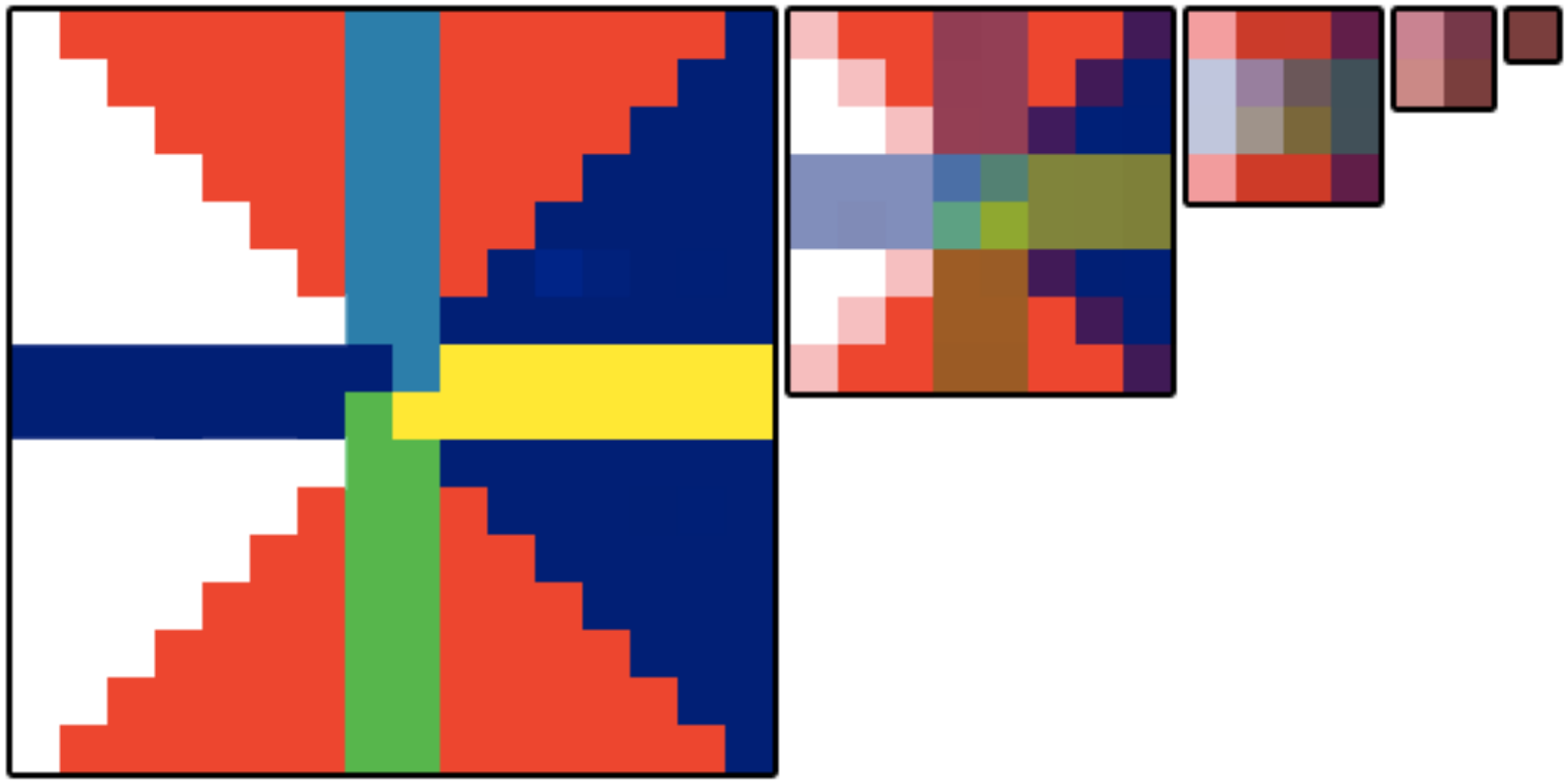


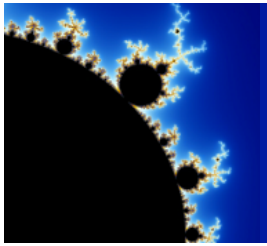
Mipmapped Textures

- Mipmapping allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Computationally more efficient
- Declare mipmap level during texture definition
  - gl.texImage2D(gl.TEXTURE_*D, level, …)



Mipmapped Textures

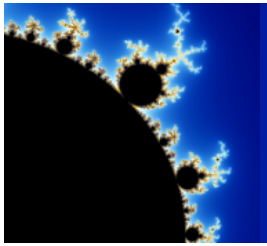




Texture filtering

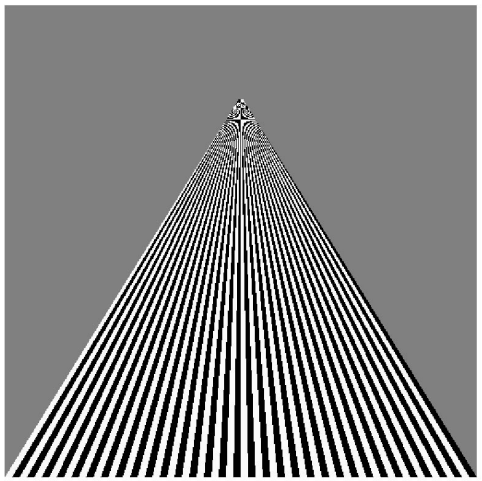
- NEAREST = choose 1 pixel from the biggest mip
- LINEAR = choose 4 pixels from the biggest mip and blend them

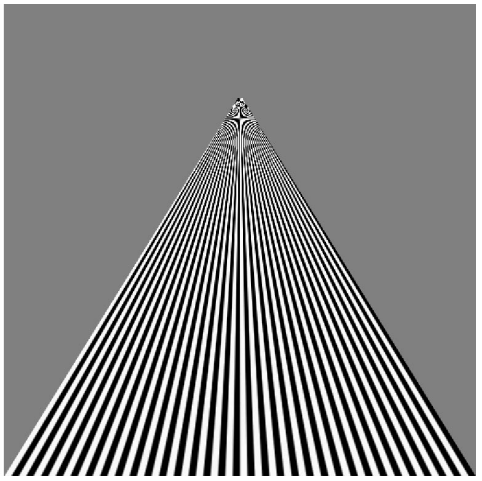
- With MinMap:
  - NEAREST_MIPMAP_NEAREST = choose the best mip, then pick one pixel from that mip
  - LINEAR_MIPMAP_NEAREST = choose the best mip, then blend 4 pixels from that mip
  - NEAREST_MIPMAP_LINEAR = choose the best 2 mips, choose 1 pixel from each, blend them
  - LINEAR_MIPMAP_LINEAR = choose the best 2 mips. choose 4 pixels from each, blend them



Example

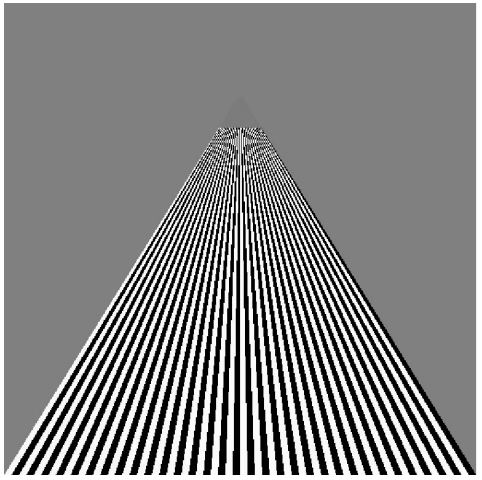
point sampling

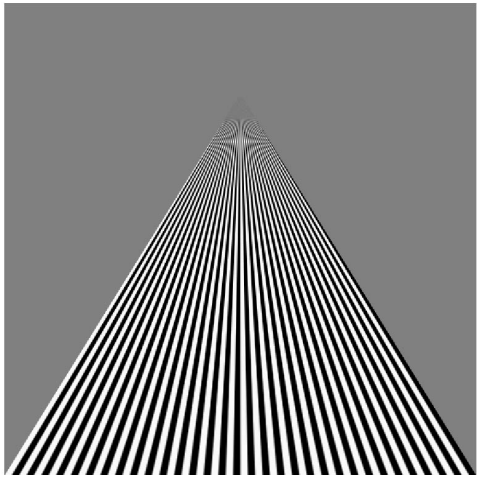




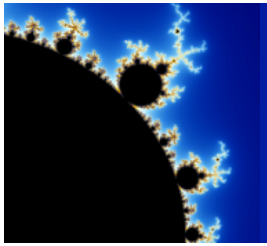
linear filtering

mipmapped point sampling



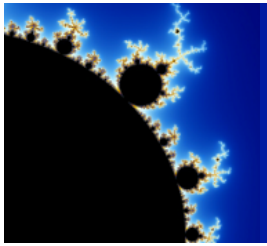


mipmapped linear filtering



Applying Textures

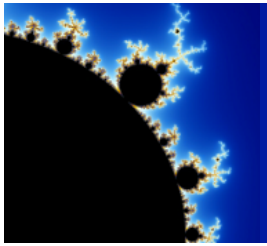
- Texture can be applied in many ways
  - texture fully determines color
  - modulated with a computed color
  - blended with an environmental color
- Can also use multiple texture units



Fragment Shader

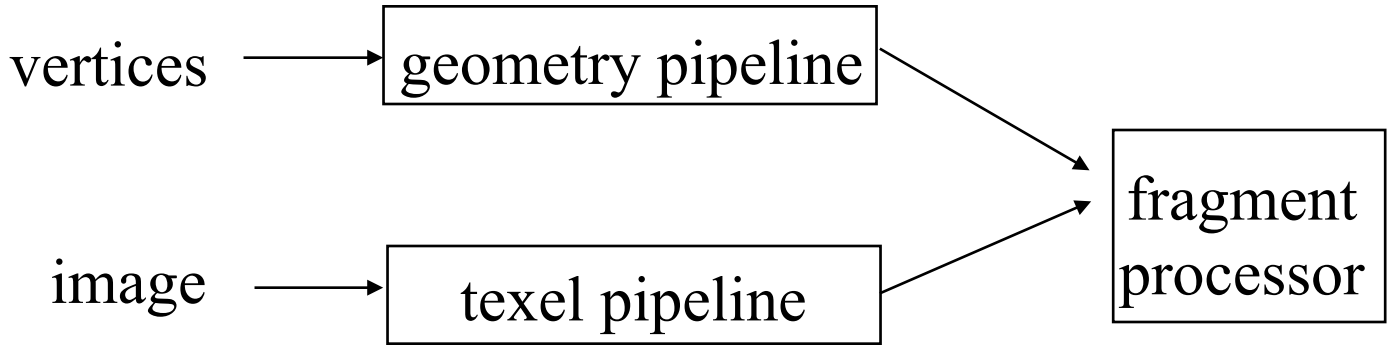
- Textures are applied in fragment shader by a **sampler**
- Samplers return a texture color from a texture object

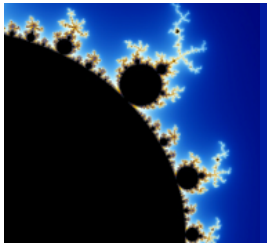
```
varying vec4 color;  //color from rasterizer
varying vec2 texCoord; //texture coordinate from rasterizer
uniform sampler2D texture; //texture object from application

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



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

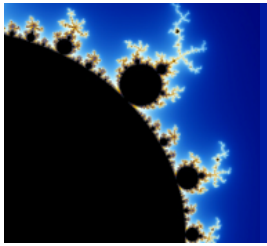




Vertex Shader

- 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

```
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
```



Linking with Shaders

```
var vTexCoord = gl.getAttribLocation( program, "vTexCoord" );
gl.enableVertexAttribArray( vTexCoord );
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0);

// 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
//   gl.activeTexture().

gl.uniform1i( glGetUniformLocation(program, "texture"), 0 );
```