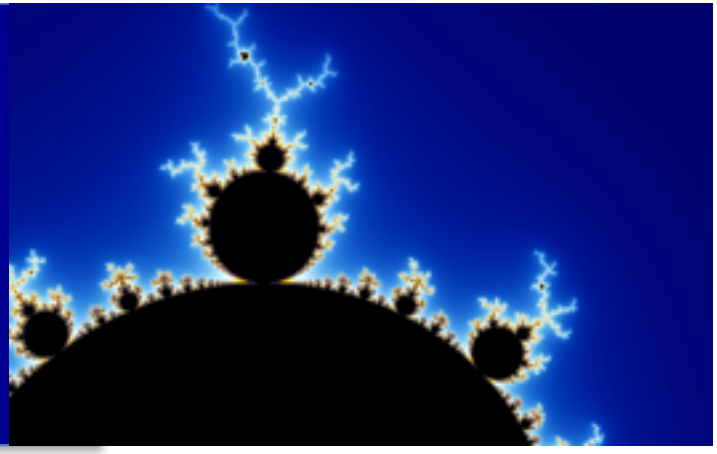
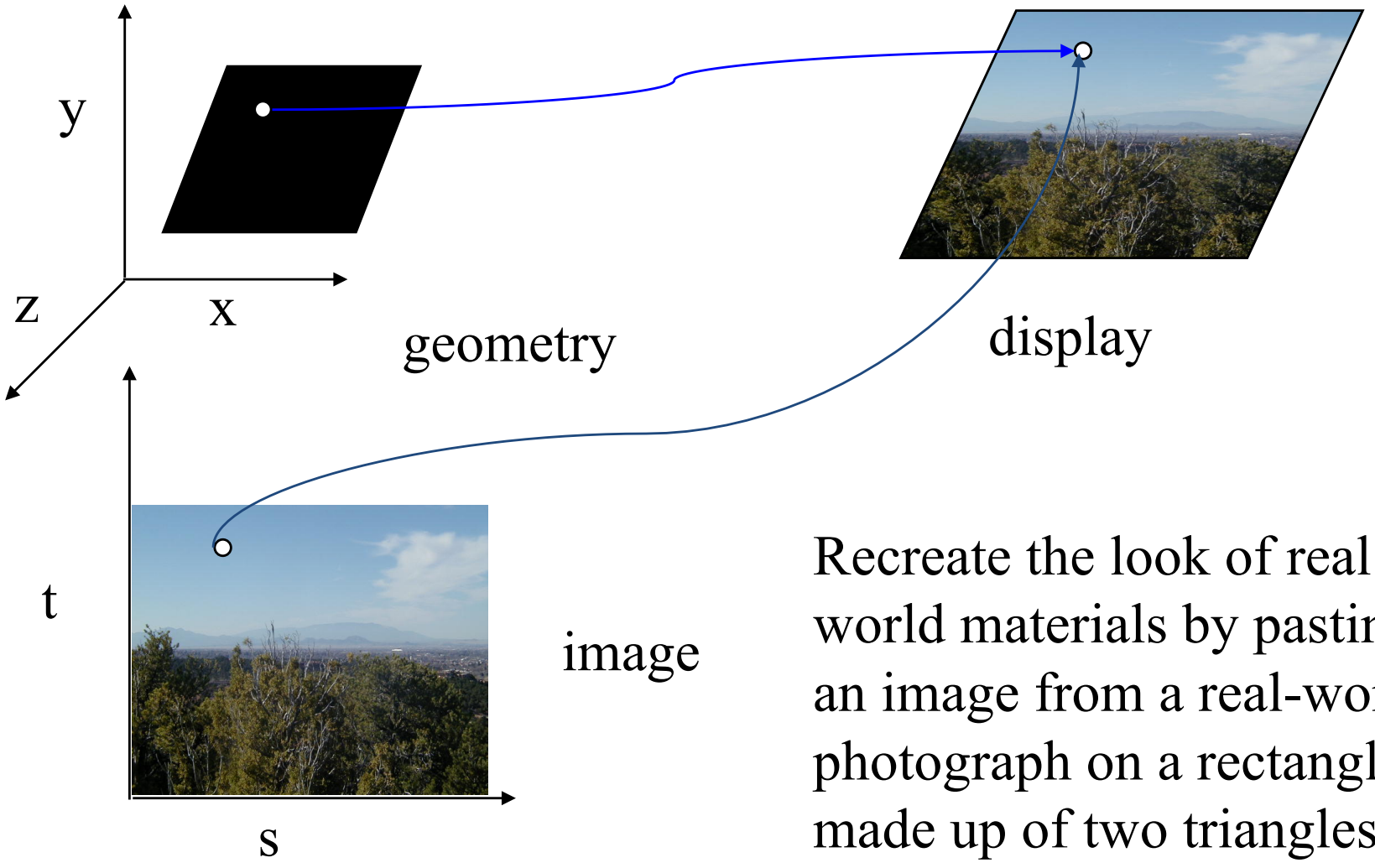
Computer Graphics



WebGL Texture Mapping

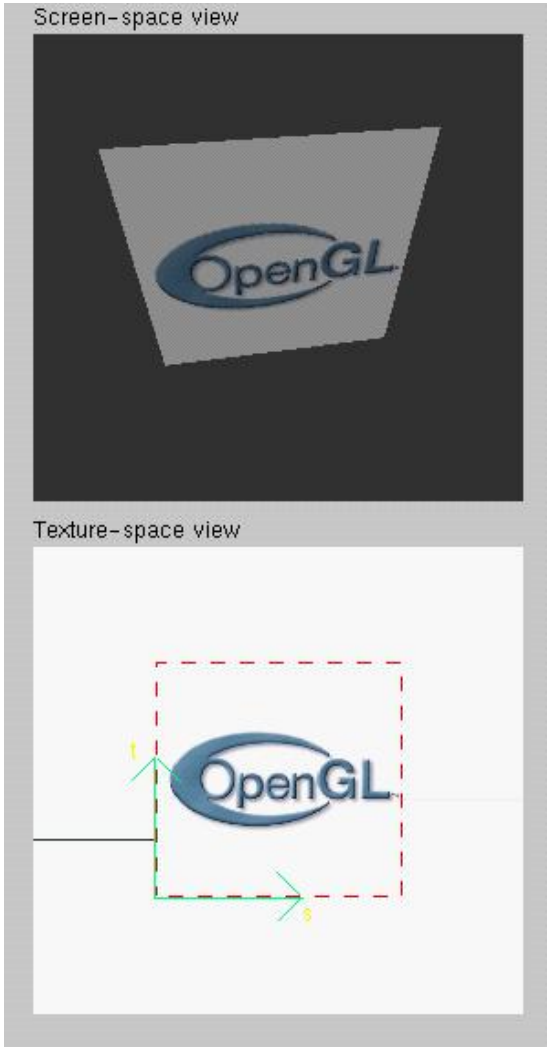
Texture Mapping



Recreate the look of real world materials by pasting an image from a real-world photograph on a rectangle made up of two triangles

Texture Example

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



Access local image files

- Start Google Chrome with local file access
  - Set the --args --allow-file-access-from-files flag
  - Mac OS

    open /Applications/Google\ Chrome.app --args --allow-file-access-from-files
  - Windows

On Windows Operating System

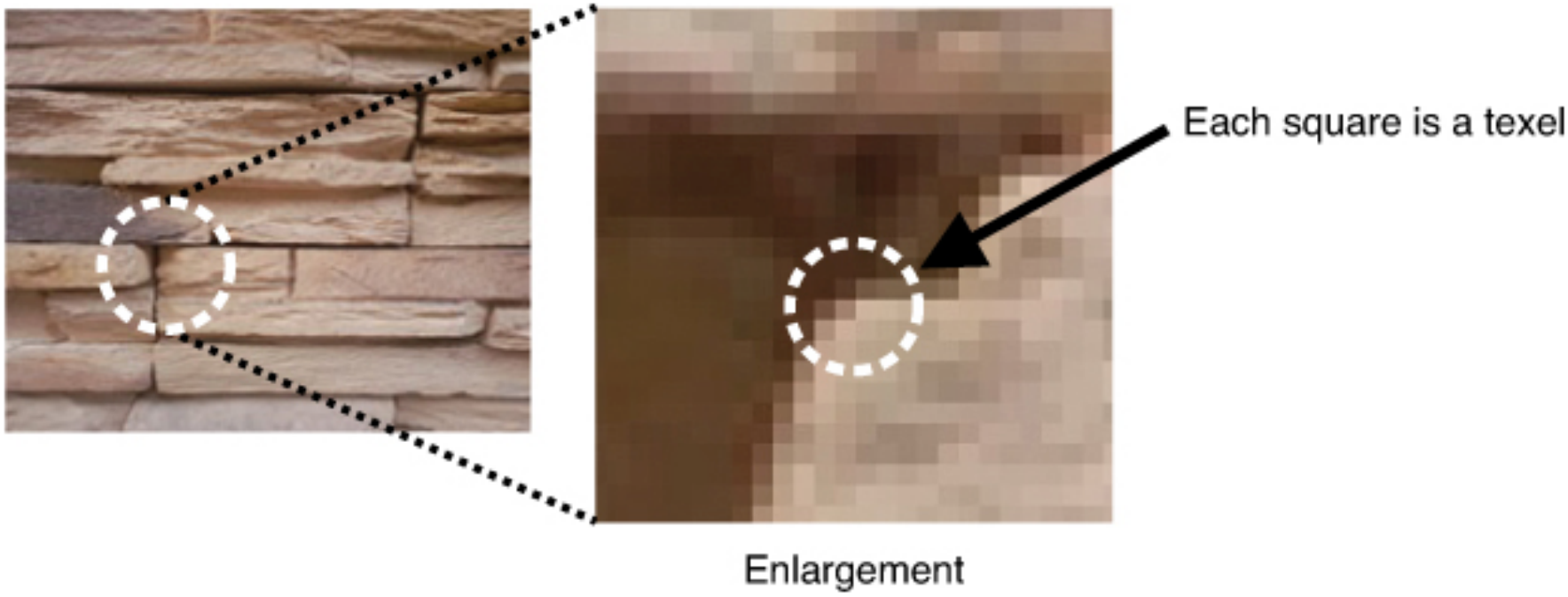
1. Get the url of your Chrome Installation **path to your chrome installation** e.g C:\Users\-your-user-name\AppData\Local\Google\Chrome\Application>

2. Launch the Google Chrome browser from the command line window with the additional argument '**–allow-file-access-from-files**'. E.g '**path to your chrome installation\chrome.exe --allow-file-access-from-files**'

3. **Temporary method you can use each time you are testing**
   - Copy the existing chrome launcher
   - Do as above and save it with a new name e.g chrome - testing
   - Alternatively, you can simply create a new launcher with the above and use it to start chrome.

Texture Mapping

The role of the texture mapping process is to assign the texture image's pixel colors to the fragments generated by the rasterization process.

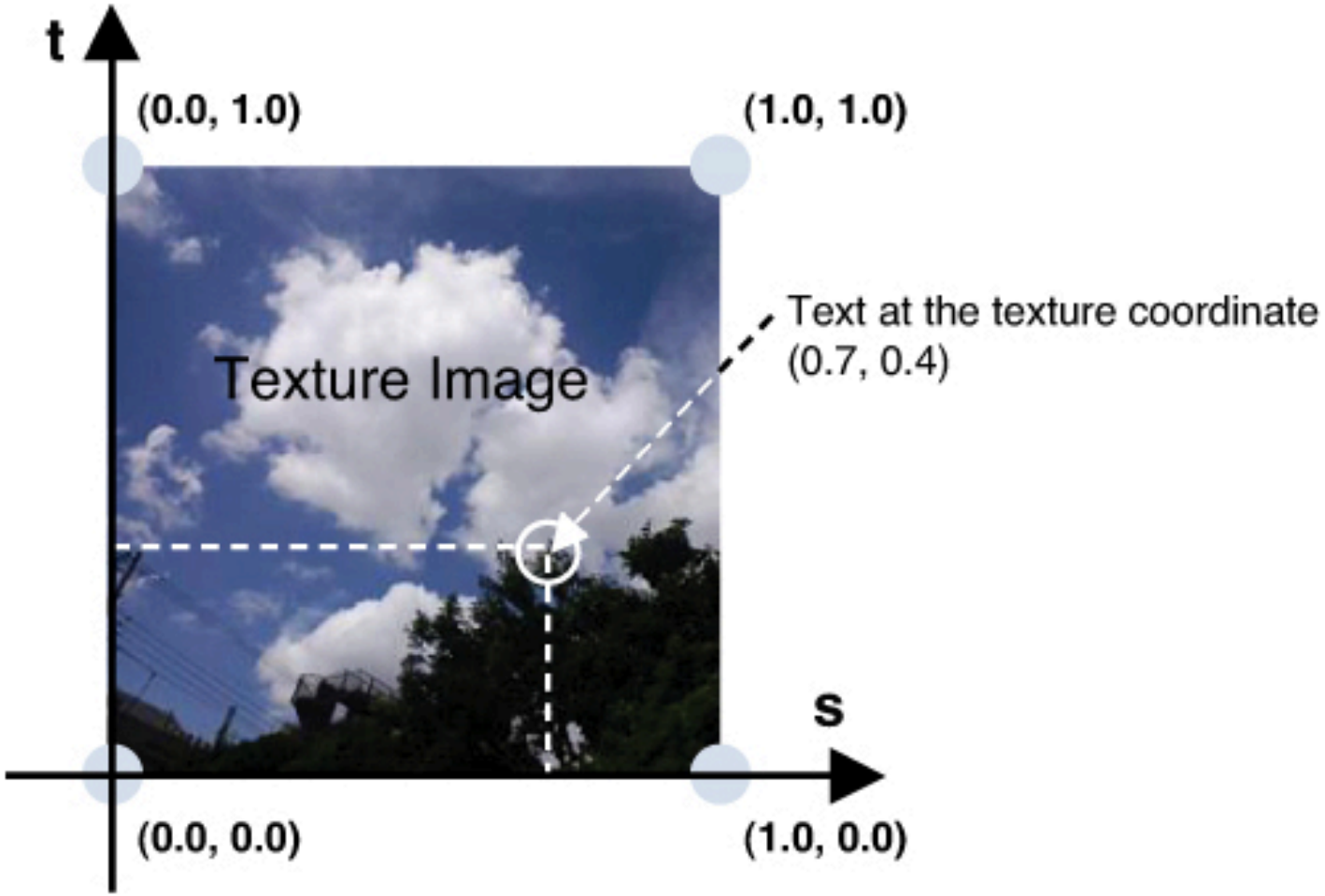


An Example

Texture image

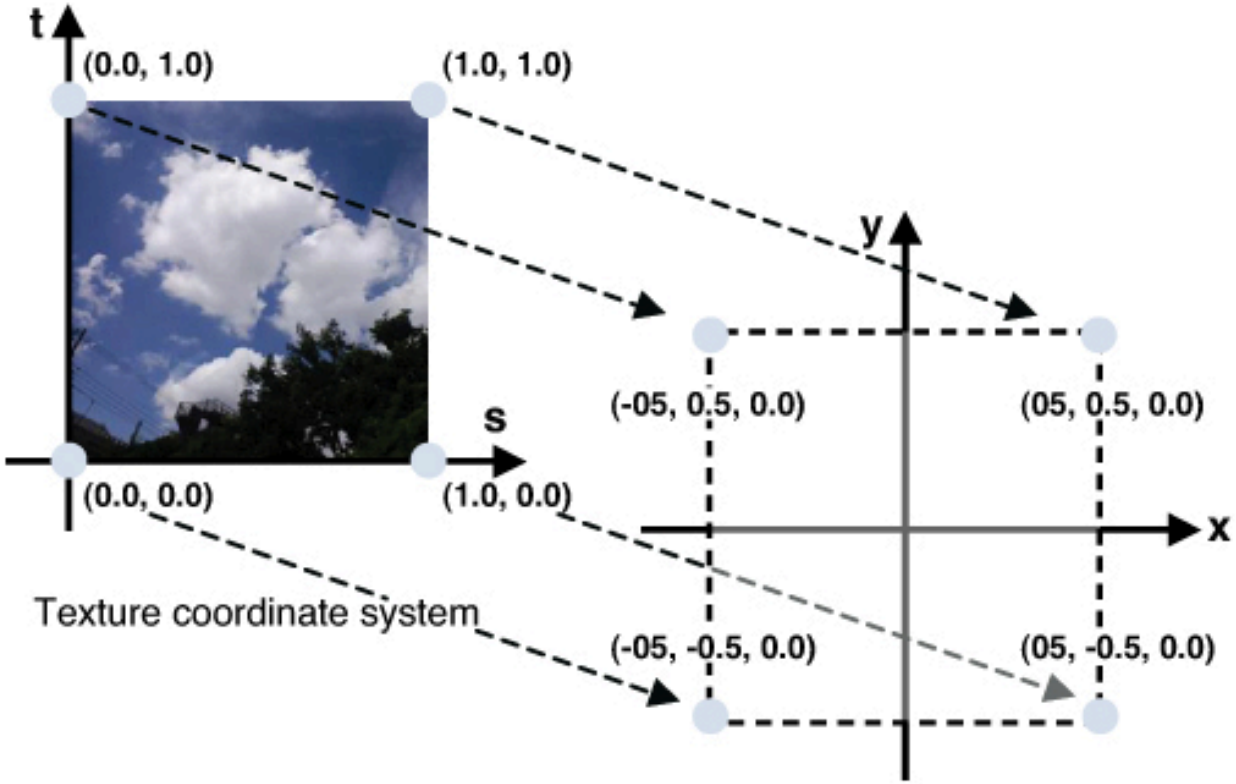
sky.jpg

WebGL Texture Coordinates



The coordinates of the four corners are not related to the image size, this allows a common approach to image handling

Pasting Texture Image onto the geometric shape

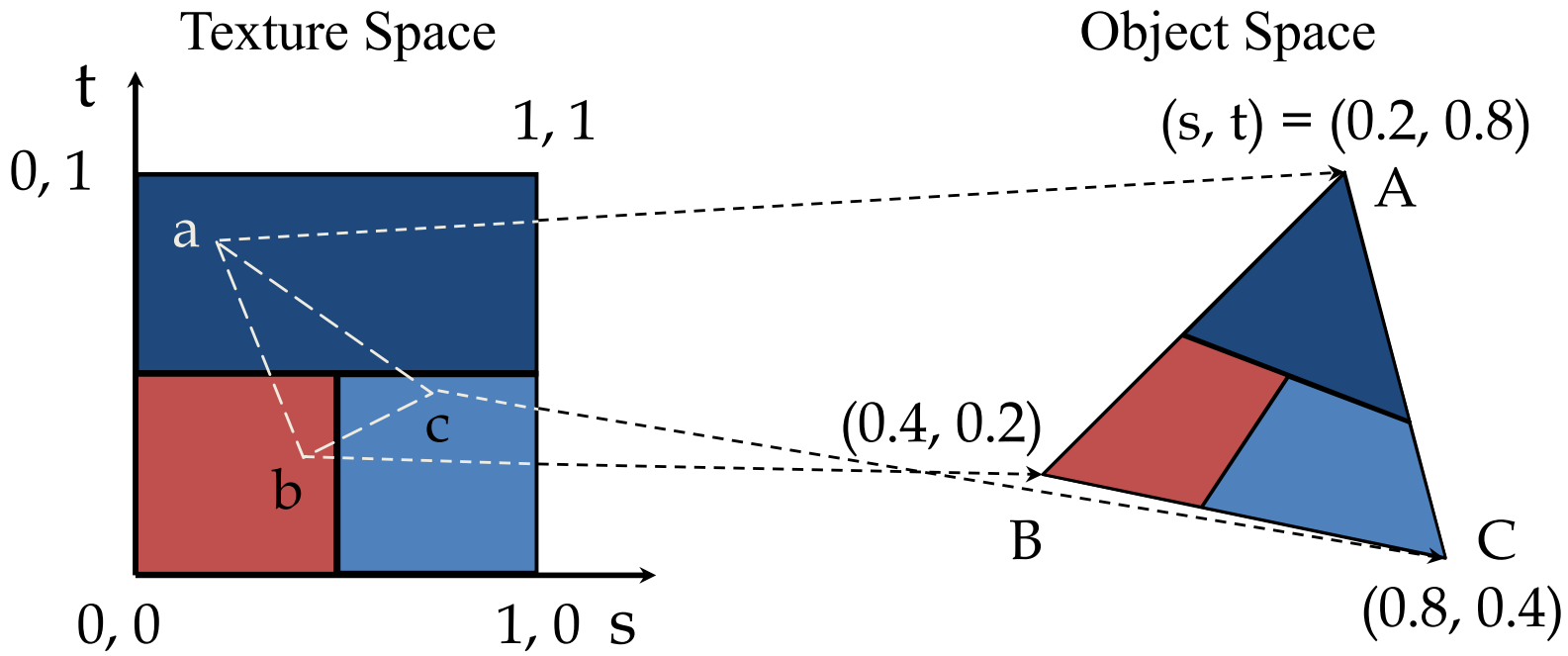


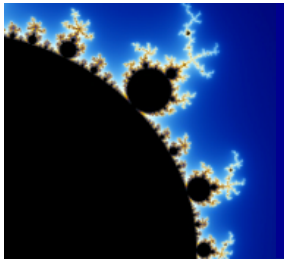
The resulting image produced by WebGL

Step 2: Mapping a Texture

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

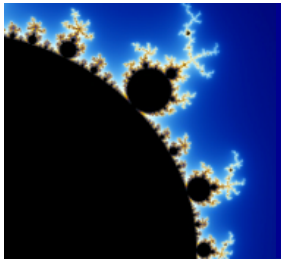
```
var texCoord = [
    vec2(0.2, 0.8),
    vec2(0.4, 0.2),
    vec2(0.8, 0.4),
];
```





Code walkthrough: “textureSquare-image”

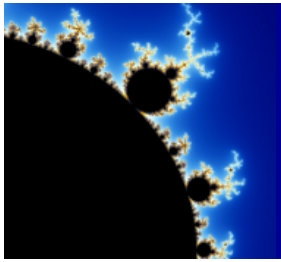
- Part 1: Receive the texture coordinates in the vertex shader and then pass them to the fragment shader.
- Part 2: Paste the texture image onto the geometric shape inside the fragment shader.
- Part 3: Set the texture coordinates (initVertexBuffers()).
- Part 4: Prepare the texture image for loading, and request the browser to read it.
- Part 5: Configure the loaded texture so that it can be used in WebGL (loadTexture()).



Part 3: Set the texture coordinates

```
// vertex positions
/*  A  C        C
    B         B D   */
var vertices = [
    vec2( -0.5, 0.5),
    vec2( -0.5,  -0.5),
    vec2( 0.5, 0.5),
    vec2( -0.5,  -0.5),
    vec2( 0.5, 0.5),
    vec2( 0.5,  -0.5),
];

// texture coordinates
// version 1 -- basic ve
/*  A C      C
    B     B D */
var texCoord = [
    vec2(0, 1),
    vec2(0, 0),
    vec2(1, 1),

    vec2(0, 0),
    vec2(1, 1),
    vec2(1, 0),
];
```

```
function InitVertexBuffers()
{
    // position buffer
    var vBuffer = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

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

    // texture buffer
    var tBuffer = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, tBuffer );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoord), gl.STATIC_DRAW );

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

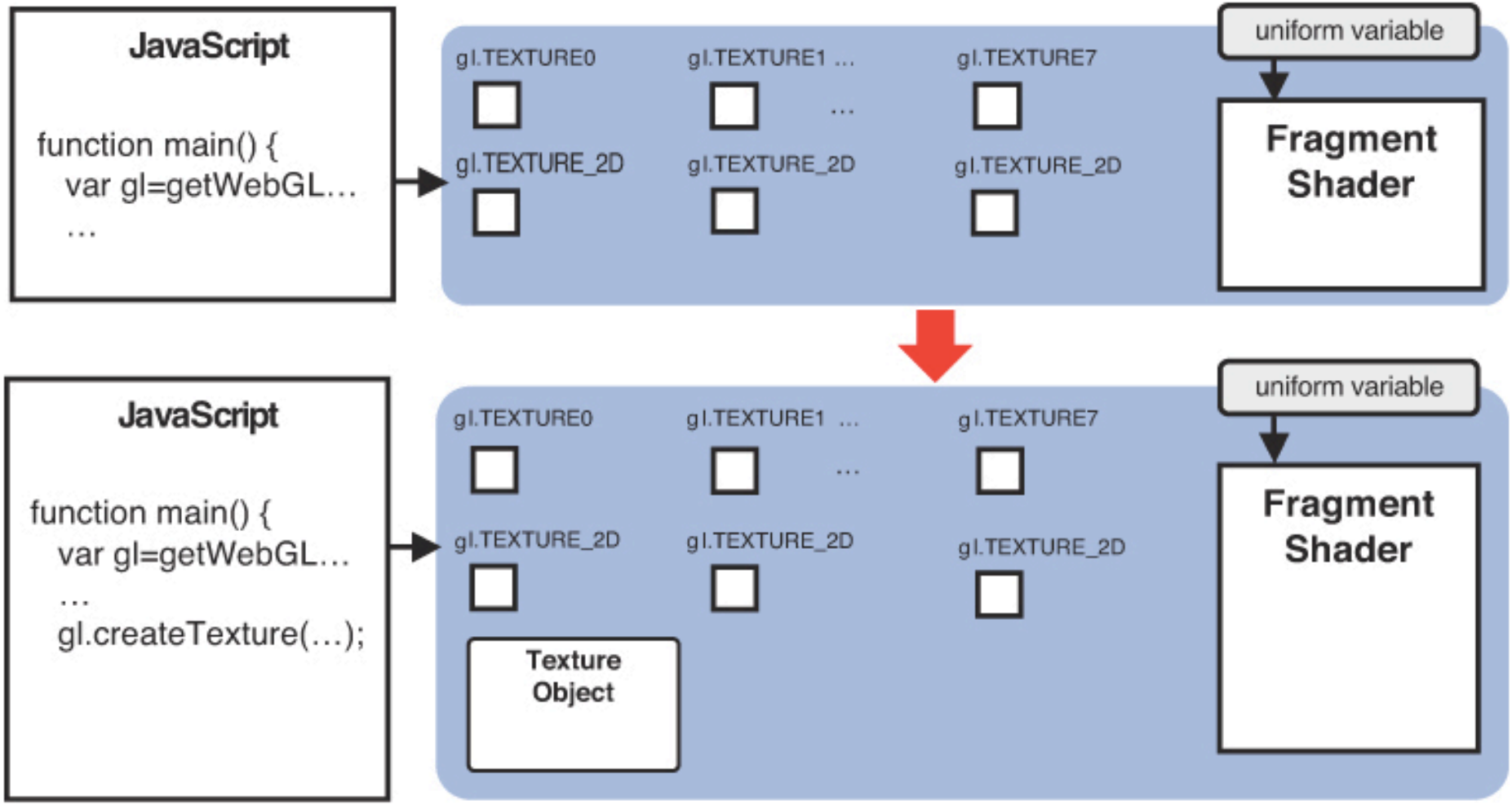


Part 4: Setting up and loading images

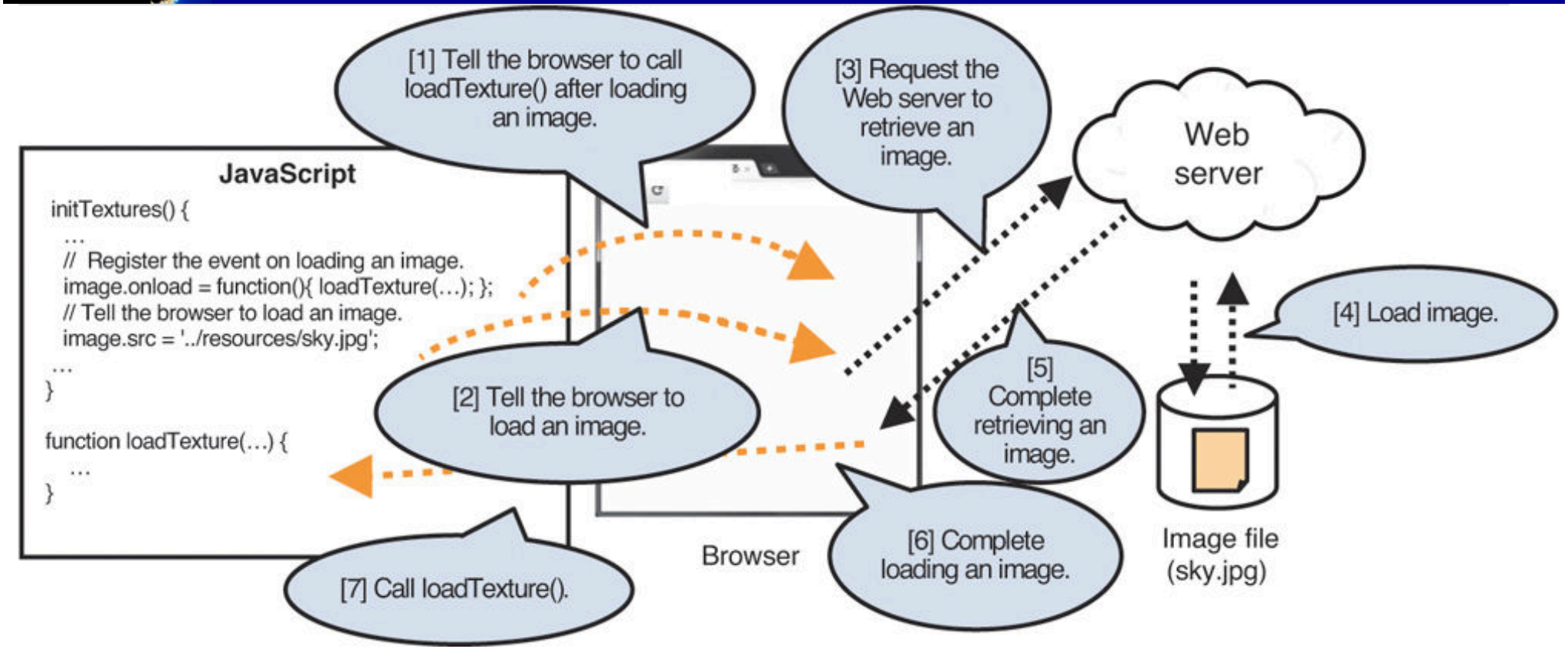
```
// ========  Establish Textures ==================
// create the texture object
texture = gl.createTexture();

// create the image object
texture.image = new Image();

// register the event handler to be called on loading an image
texture.image.onload = function() {  loadTexture(texture);}

// Tell the broswer to load an image
texture.image.src='sky.jpg';
```

gl.createTexture()

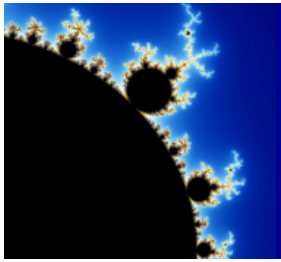
Creates a texture object in WebGL for managing an image



Asynchronous loading texture images



For WebGL, it is necessary to read images indirectly by requesting the browser to do it.



Part 5: Setting up and loading images

```
function loadTexture(texture)
{
    // Flip the image's y axis
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);

    // Enable texture unit 0
    gl.activeTexture(gl.TEXTURE0);

    // bind the texture object to the target
    gl.bindTexture( gl.TEXTURE_2D, texture );

    // set the texture image
    gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, texture.image );
    // v1 (needed combination for images that are not powers of 2
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);

    // set the texture parameters
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST );
    gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST );

    // set the texture unit 0 the sampler
    gl.uniform1i(gl.getUniformLocation(program, "texture"), 0);
}
```

Flipping an Image's Y-axis

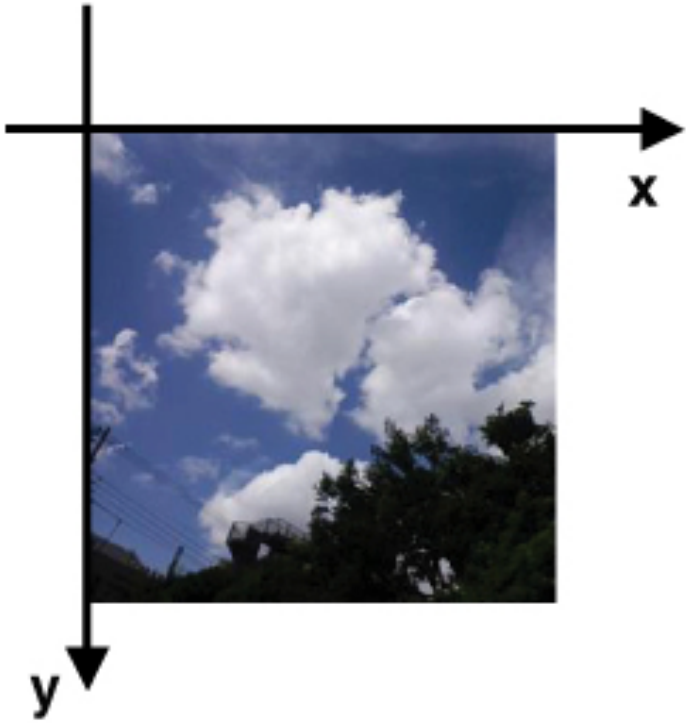
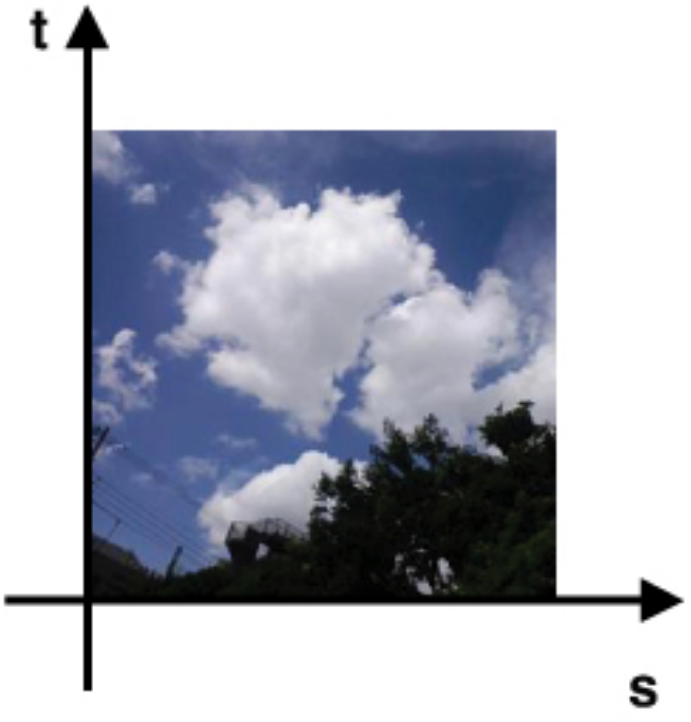
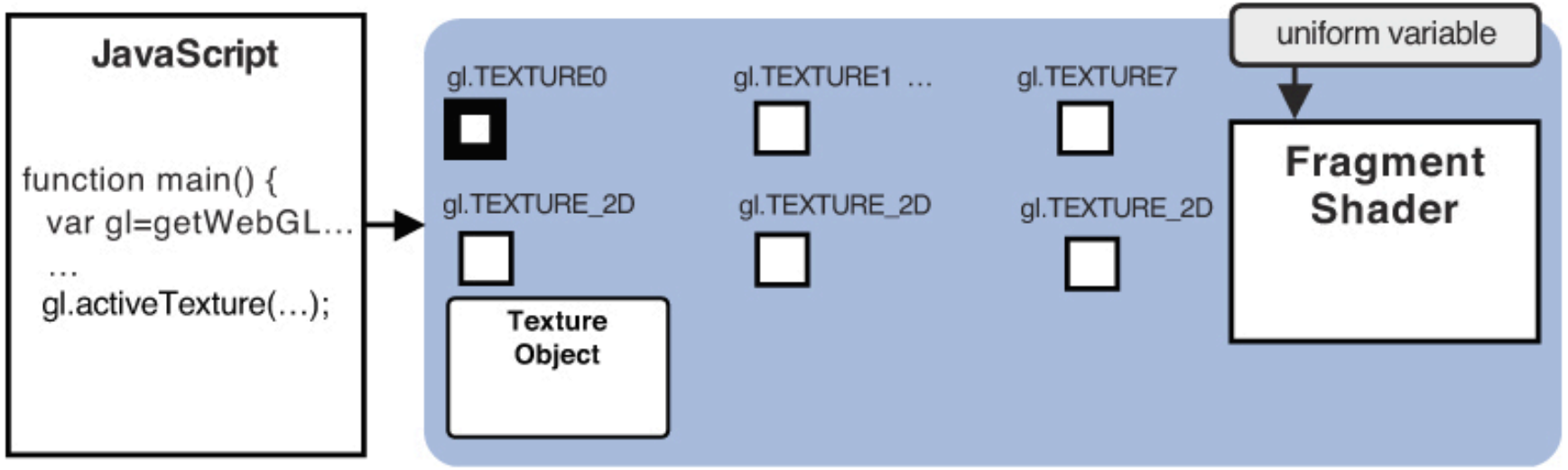


Image coordinate system



WebGL coordinate system

Active texture unit



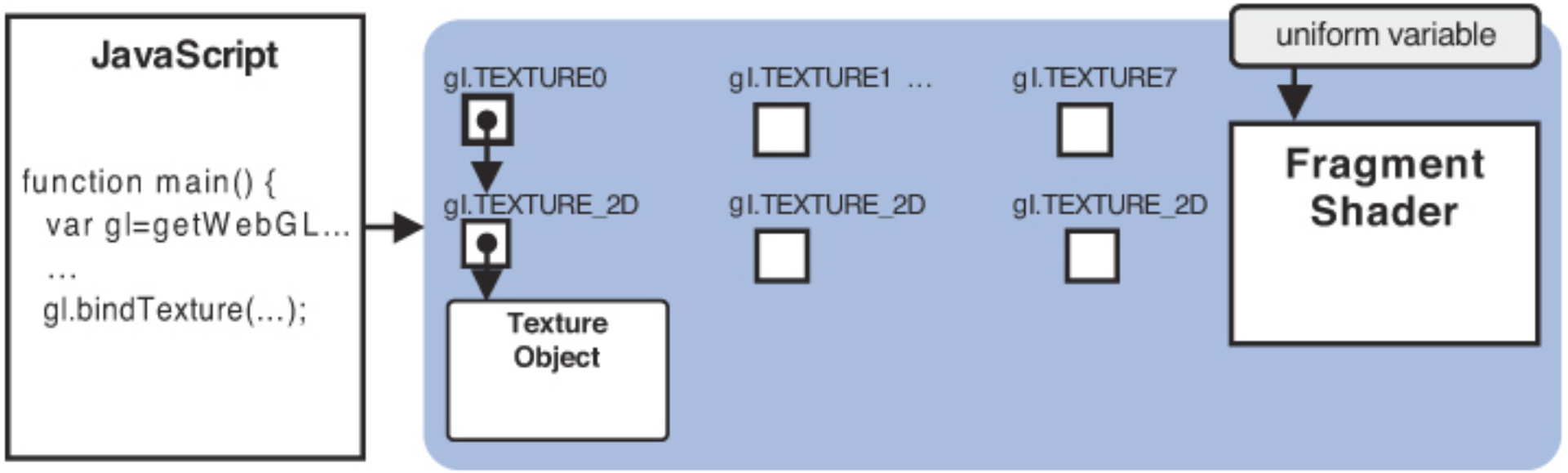
gl.activeTexture(gl.TEXTURE0);

`gl.activeTexture(texUnit)`

Make the texture unit specified by *texUnit* active.

Parameters	texUnit	Specifies the texture unit to be made active: `gl.TEXTURE0`, `gl.TEXTURE1`, ..., or `gl.TEXTURE7`. The tailing number indicates the texture unit number.
Return value	None	
Errors	INVALID_ENUM:	*texUnit* is none of these values

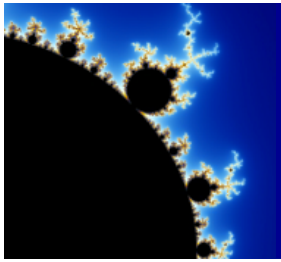
Bind a texture object to the target



gl.bindTexture(gl.TEXTURE_2D, texture);

`gl.bindTexture(target, texture)`

Enable the texture object specified by *texture* and bind it to the *target*. In addition, if a texture unit was made active by `gl.activeTexture()`, the texture object is also bound to the texture unit.

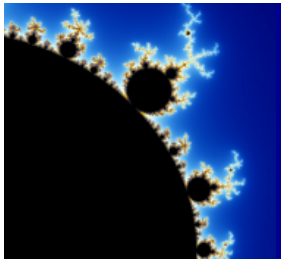


gl.texParameteri

`gl.texParameteri(target, pname, param)`

Set the value specified by *param* to the texture parameter specified by *pname* in the texture object bound to *target*.

Parameters	target	Specifies `gl.TEXTURE_2D` or `gl.TEXTURE_CUBE_MAP`.
	pname	Specifies the name of the texture parameter (Table 5.3).
	param	Specifies the value set to the texture parameter *pname* (Table 5.4, Table 5.5).
Return value	None	
Errors	INVALID_ENUM	*target* is none of the preceding values
	INVALID_OPERATION	no texture object is bound to *target*



Texture Parameters

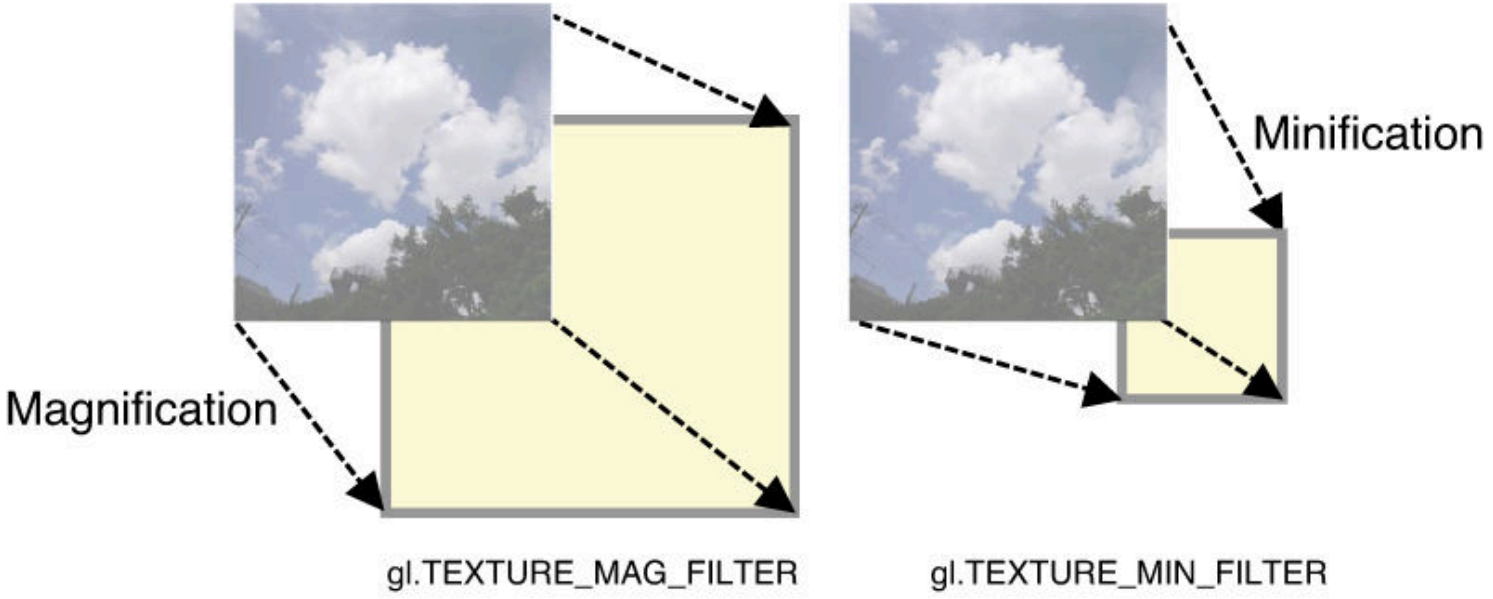
- **Magnification method** (`gl.TEXTURE_MAG_FILTER`): The method to magnify a texture image when you map the texture to a shape whose drawing area is larger than the size of the texture. For example, when you map a 16×16 pixel image to a 32×32 pixel shape, the texture should be doubled in size. WebGL needs to fill the gap between texels due to the magnification, and this parameter specifies the method used to fill the gap.

- **Minification method** (`gl.TEXTURE_MIN_FILTER`): The method of minifying a texture image when you map the texture to a shape whose drawing area is smaller than the size of the texture. For example, when you map a 32×32 pixel image to a 16×16 pixel shape, the texture should be reduced in size. To do that, the system needs to cull texels to fit the target size. This parameter specifies the method used to cull texels.

- **Wrapping method on the left and right side** (`gl.TEXTURE_WRAP_S`): How to fill the remaining regions on the left side and the right side of a subregion when you map a texture image to the subregion of a shape.

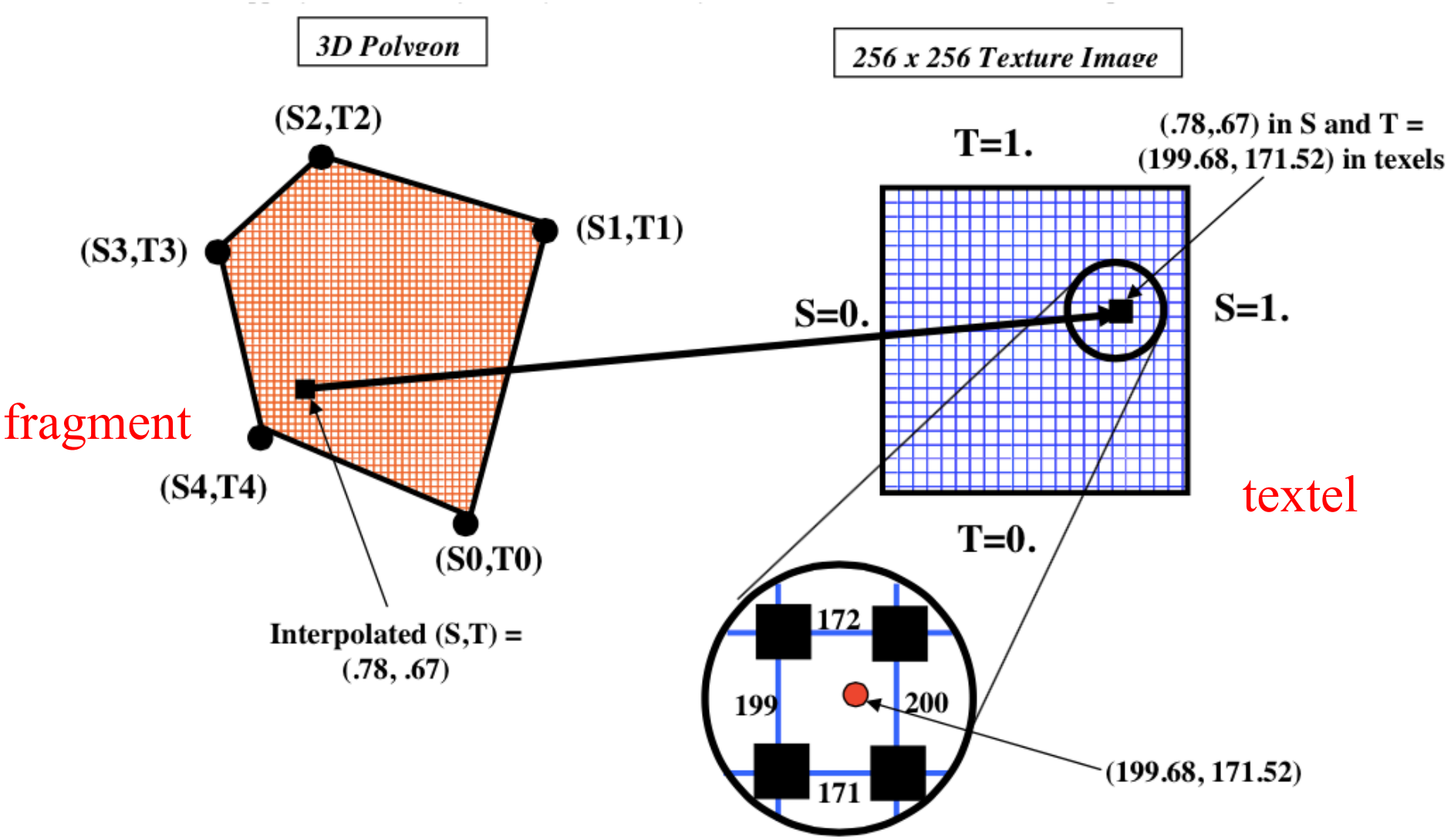
- **Wrapping method on top and bottom** (`gl.TEXTURE_WRAP_T`): Similar to (3), the method used to fill the remaining regions in the top and bottom of a subregion.

Magnification and Minification



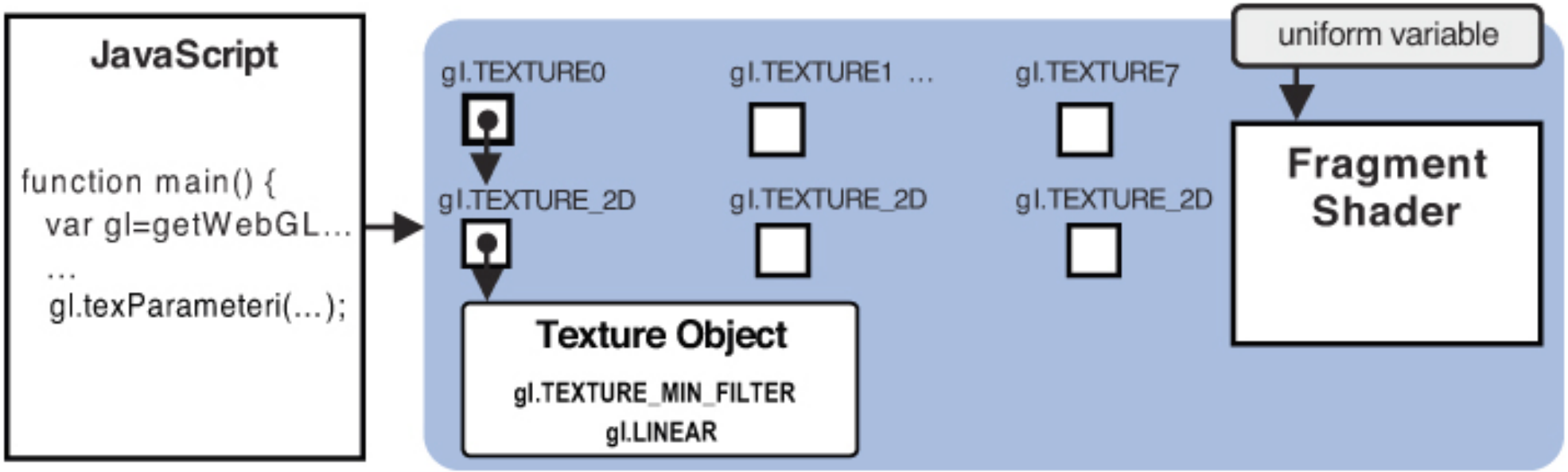
Value	Description
`gl.NEAREST`	Uses the value of the texel that is nearest (in Manhattan distance) the center of the pixel being textured.
`gl.LINEAR`	Uses the weighted average of the four texels that are nearest the center of the pixel being textured. (The quality of the result is clearer than that of `gl.NEAREST`, but it takes more time.)

Texture Mapping



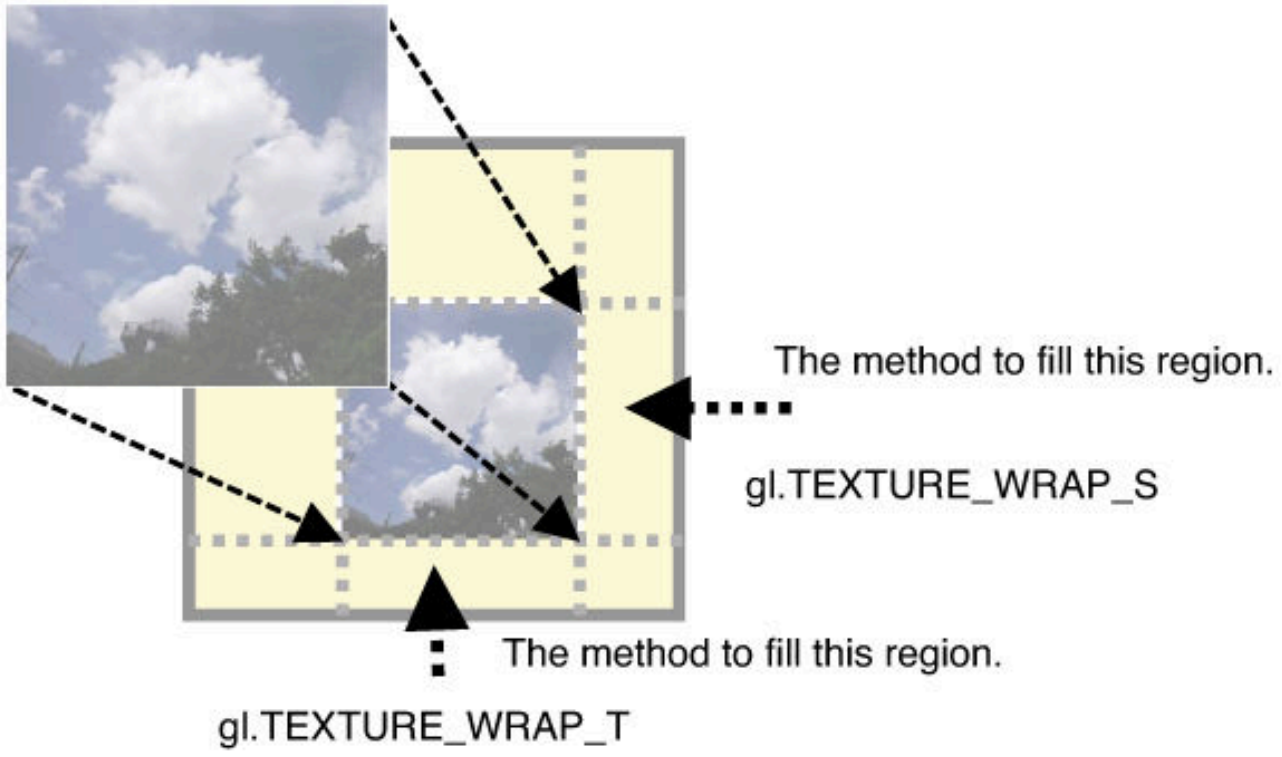
gl.texParameteri

Texture Parameter	Description	Default Value
gl.TEXTURE_MAG_FILTER	Texture magnification	gl.LINEAR
gl.TEXTURE_MIN_FILTER	Texture minification	gl.NEAREST_MIPMAP_LINEAR
gl.TEXTURE_WRAP_S	Texture wrapping in s-axis	gl.REPEAT
gl.TEXTURE_WRAP_T	Texture wrapping in t-axis	gl.REPEAT



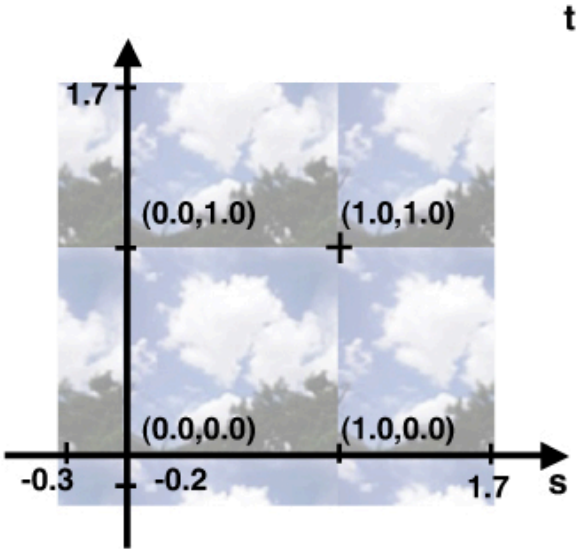
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

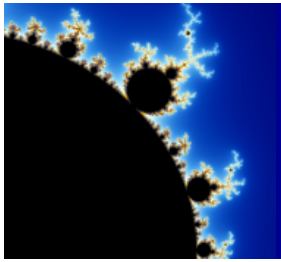
gl.TEXTURE_WRAP_T and gl.TEXTURE_WRAP_S



Value	Description
`gl.REPEAT`	Use a texture image repeatedly
`gl.MIRRORED_REPEAT`	Use a texture image mirrored-repeatedly
`gl.CLAMP_TO_EDGE`	Use the edge color of a texture image

gl.TEXTURE_WRAP_T and gl.TEXTURE_WRAP_S





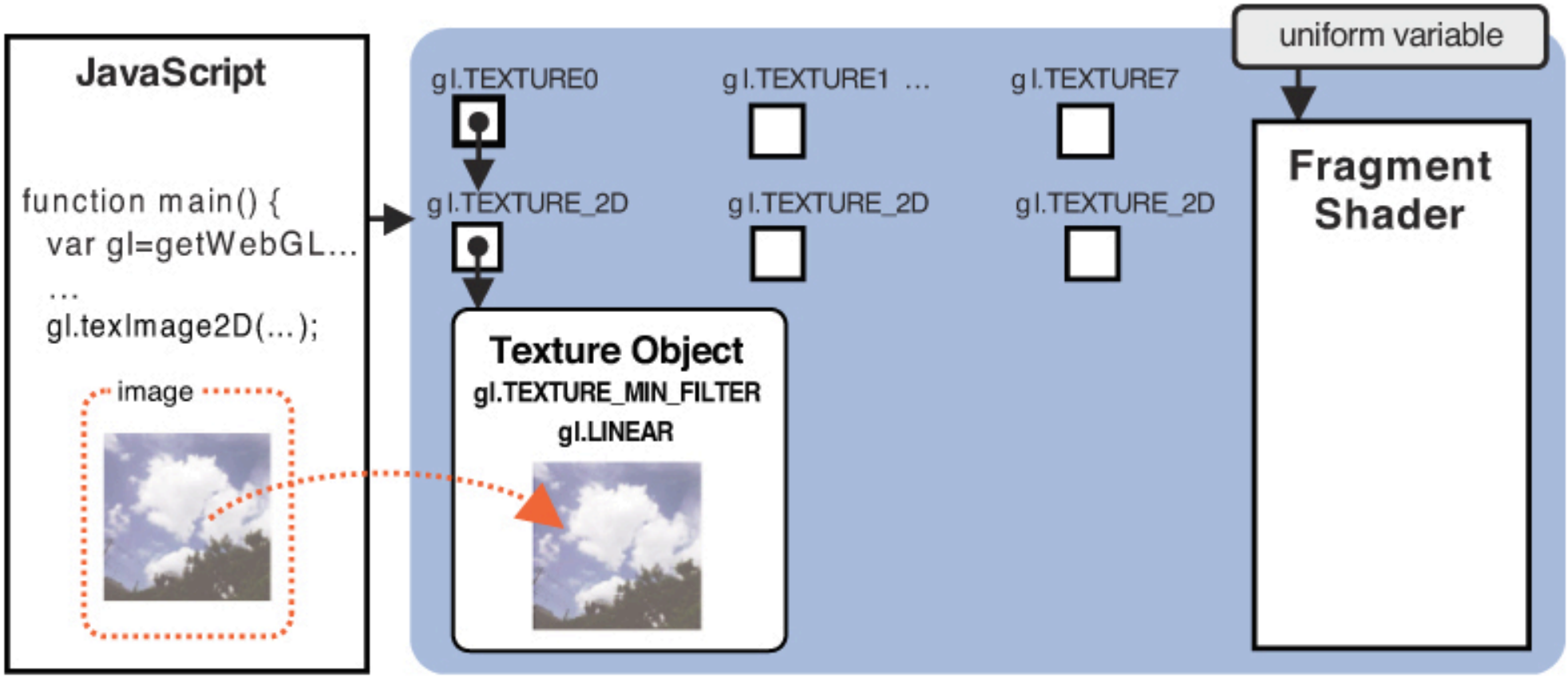
gl.texImage2D

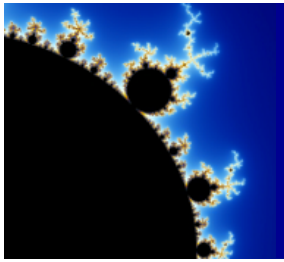
```
gl.texImage2D(target, level, internalformat, format, type, image)
```

Set the image specified by *image* to the texture object bound to *target*.

Parameters	target	Specifies `gl.TEXTURE_2D` or `gl.TEXTURE_CUBE_MAP`.
	level	Specified as 0. (Actually, this parameter is used for a MIPMAP texture, which is not covered in this book.)
	internalformat	Specifies the internal format of the image (Table 5.6).
	format	Specifies the format of the texel data. This must be specified using the same value as *internalformat*.
	type	Specifies the data type of the texel data (Table 5.7).
	image	Specifies an Image object containing an image to be used as a texture.
Return value	None	
Errors	INVALID_ENUM	*target* is none of the above values.
	INVALID_OPERATION	No texture object is bound to *target*

gl.texImage2D

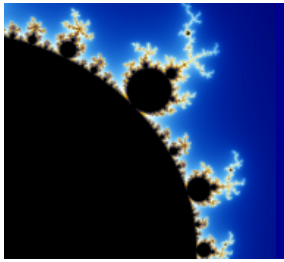




Part 1: Vertex Shader

- 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
```



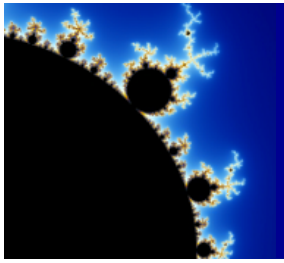
Linking with Shaders

To link with vertex shader:

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

To link with fragment shader:

```
// 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 );
```



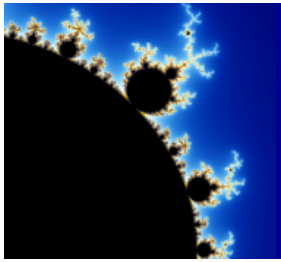
Part 2 Fragment Shader

- 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 = texture2D( texture, texCoord );
}
```

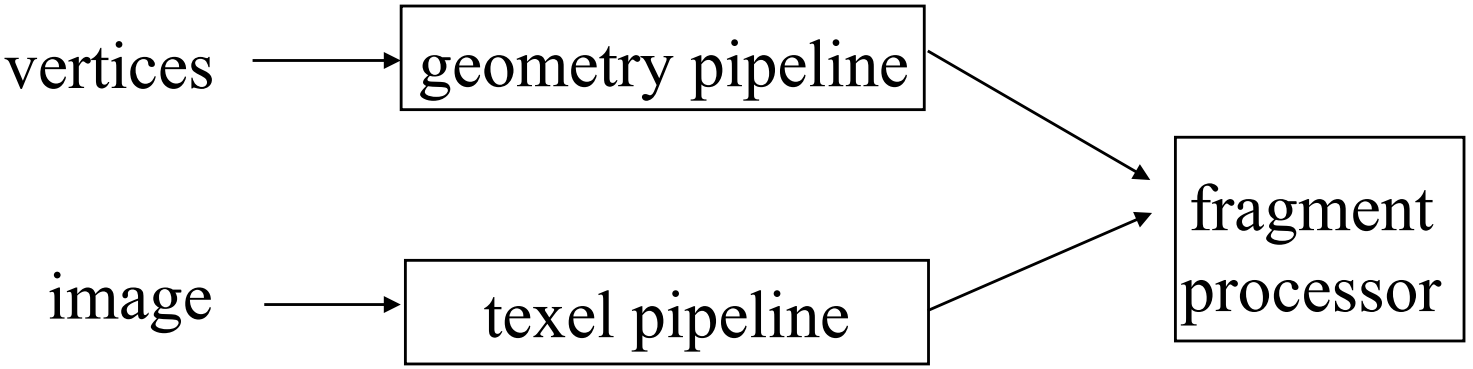
or

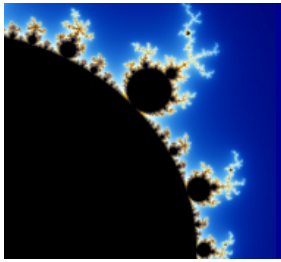
```
void main()  {
    gl_FragColor = color * texture2D( texture, texCoord );
}
```



Texture Mapping and the WebGL Pipeline

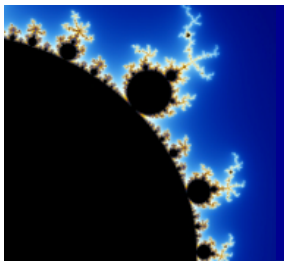
- Images and geometry flow through separate pipelines that join during fragment processing
  - "complex" textures do not affect geometric complexity



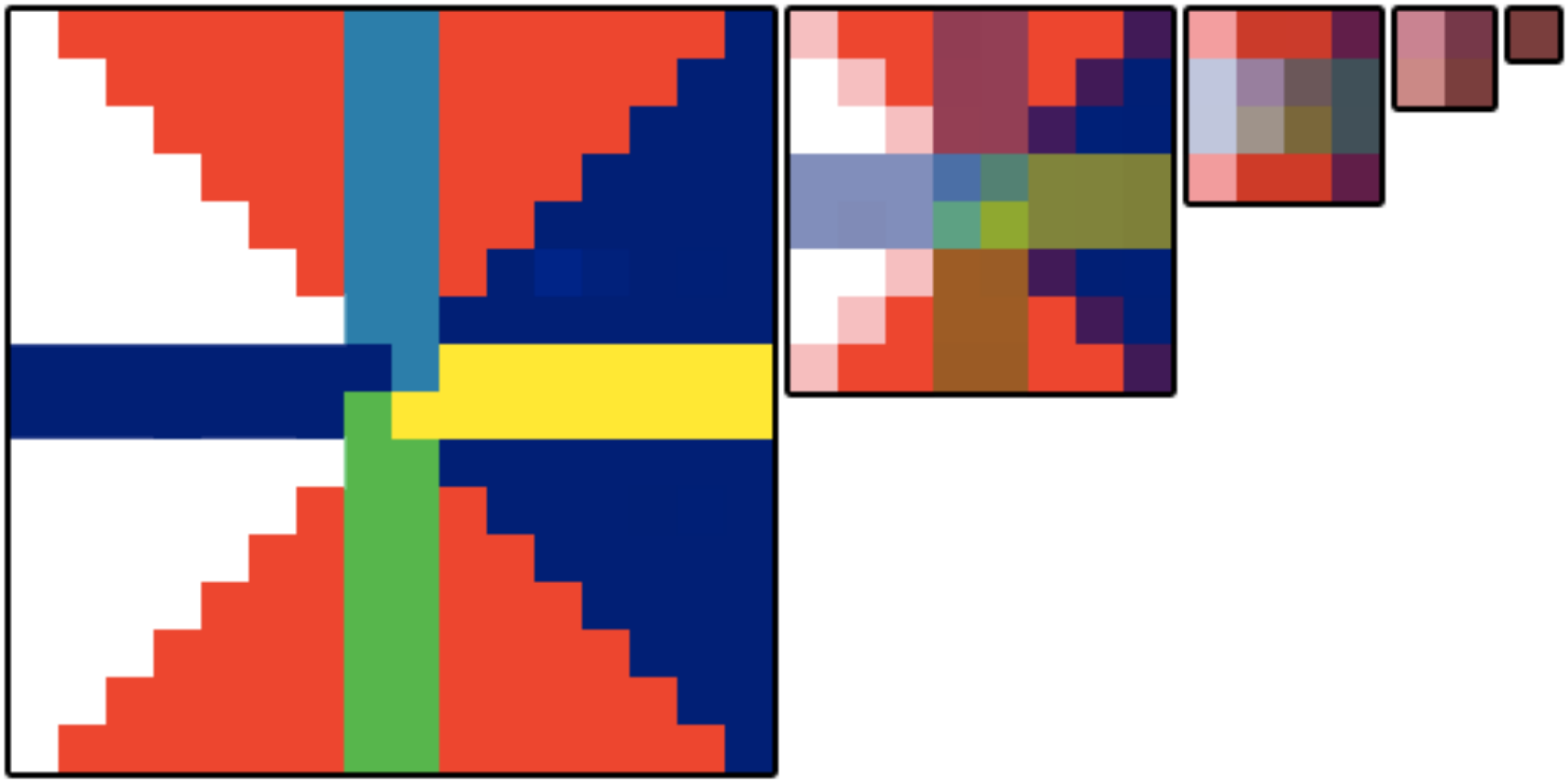


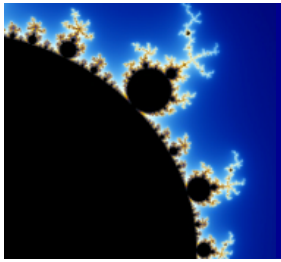
Mipmapped Textures

- 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

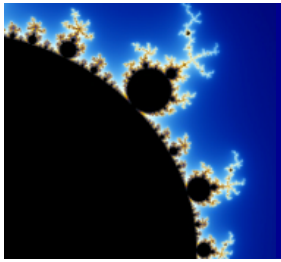




Texture filtering

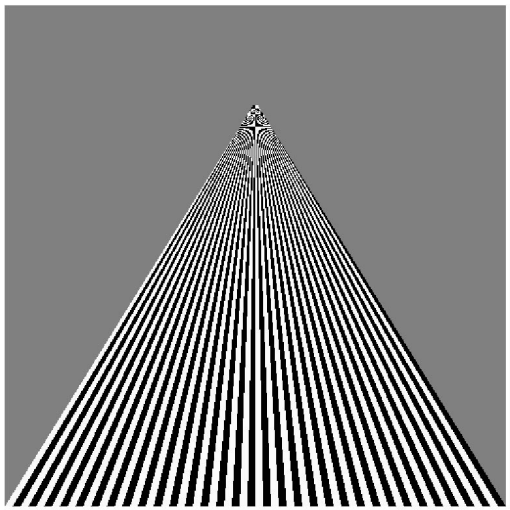
- NEAREST = choose 1 pixel from the biggest mip
- LINEAR = choose 4 pixels from the biggest mip and blend them

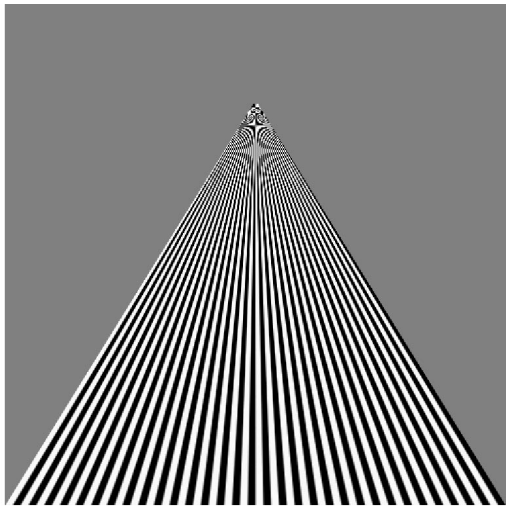
- With MinMap:
  - NEAREST_MIPMAP_NEAREST = choose the best mip, then pick one pixel from that mip
  - LINEAR_MIPMAP_NEAREST = choose the best mip, then blend 4 pixels from that mip
  - NEAREST_MIPMAP_LINEAR = choose the best 2 mips, choose 1 pixel from each, blend them
  - LINEAR_MIPMAP_LINEAR = choose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



Comparing the different schemes

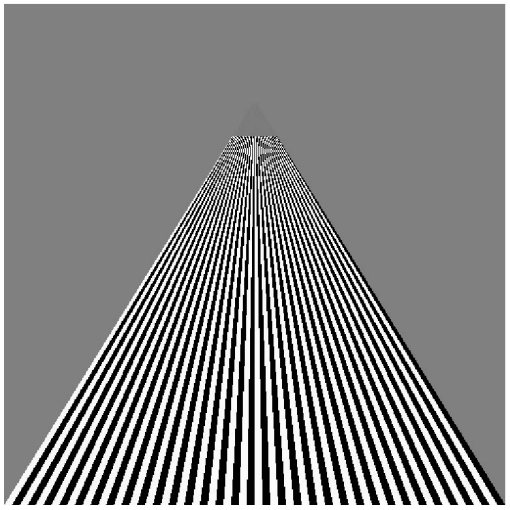
point sampling

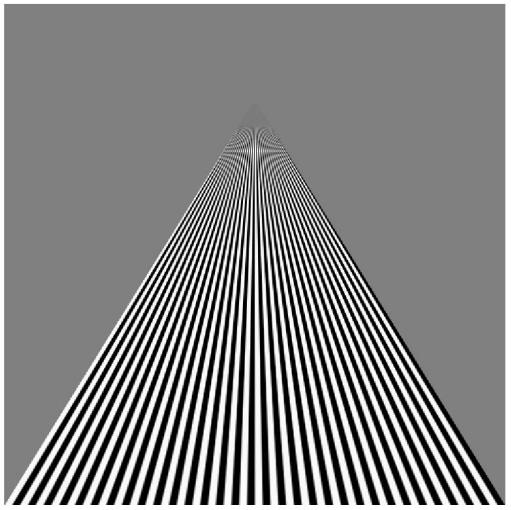




linear filtering

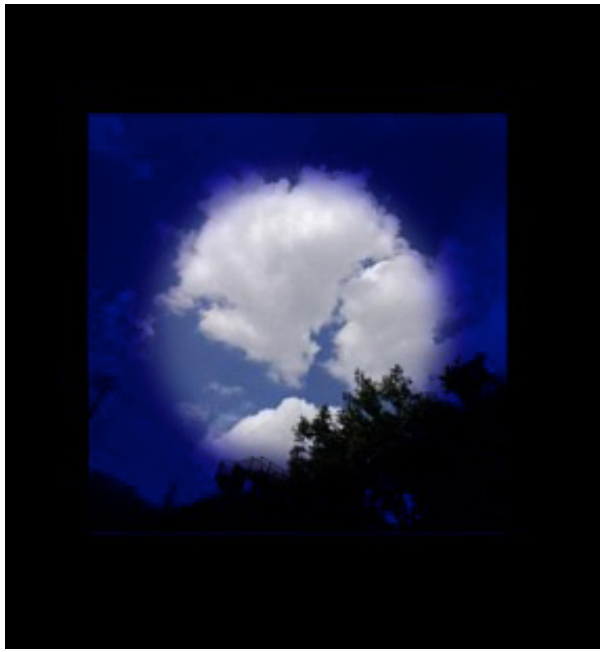
mipmapped point sampling



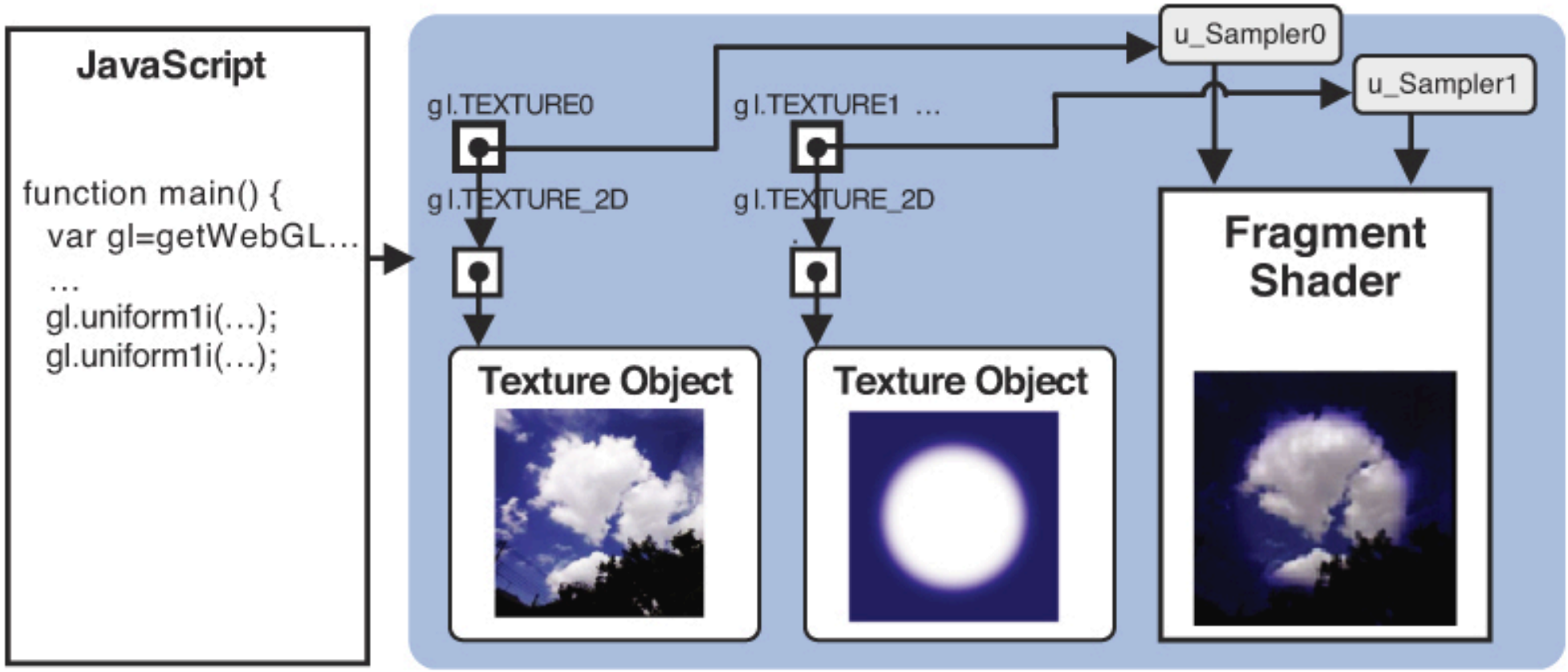


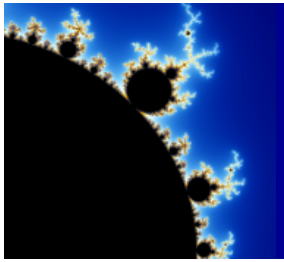
mipmapped linear filtering

Multi Texture



Multi Texture





Multiplication of two vec4 colors