COSC 414/519I: Computer Graphics

2023W2

Shan Du

2D Texture Mapping

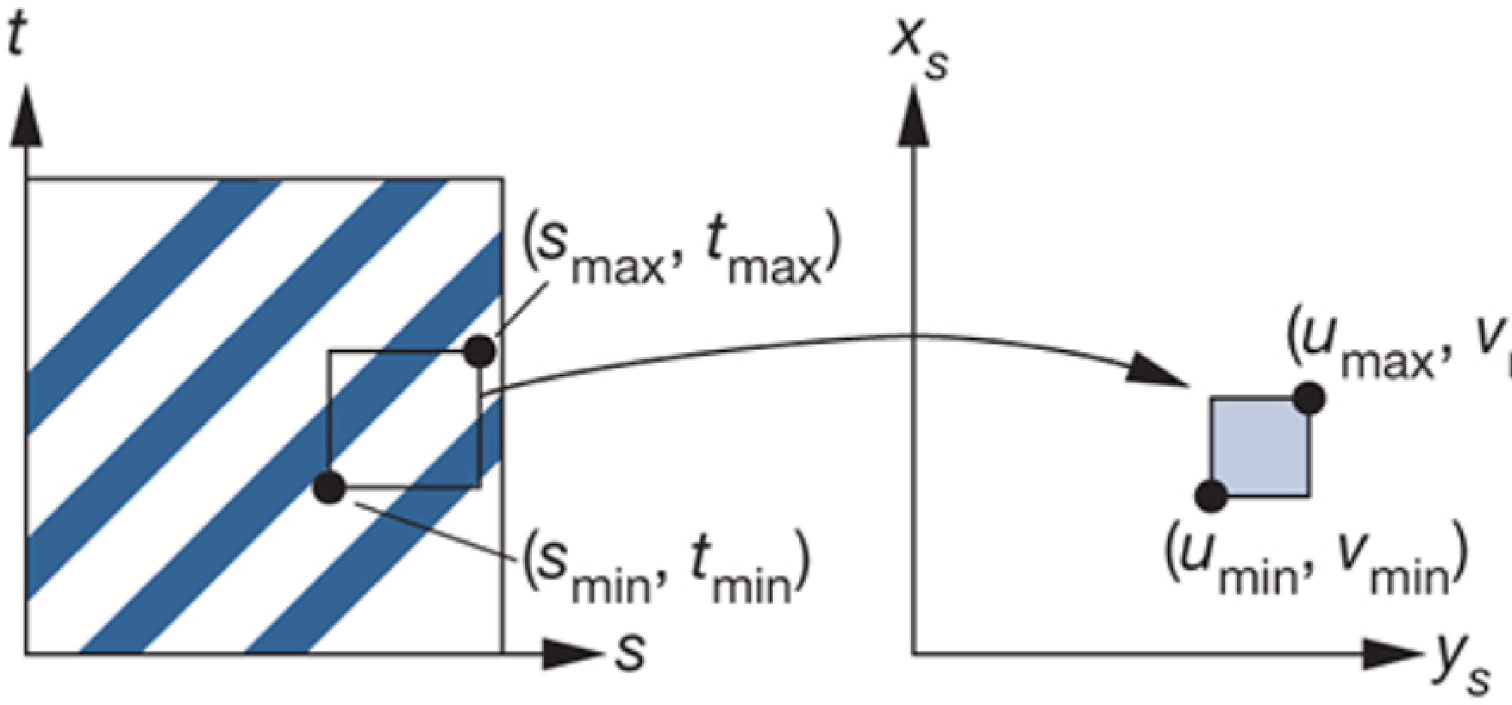
- Given a parametric surface, we can often map a point in the texture map *T(s*, *t*) to a point on the surface **p**(*u*, *v*) by a linear map of the form

$$\begin{aligned} u &= as + bt + c \\ v &= ds + et + f. \end{aligned}$$

As long as $ae \neq bd$, this mapping is invertible. Linear mapping makes it easy to map a texture to a group of parametric surface patches.

2D Texture Mapping

Figure 7.8 Linear texture mapping.



$$\begin{aligned} u &= u_{\min} + \frac{s - s_{\min}}{s_{\max} - s_{\min}}(u_{\max} - u_{\min}) \\ v &= v_{\min} + \frac{t - t_{\min}}{t_{\max} - t_{\min}}(v_{\max} - v_{\min}) . \end{aligned}$$

2D Texture Mapping

- This mapping is easy to apply, but it does not take into account the curvature of the surface. Equal-sized texture patches must be stretched to fit over the surface patch.

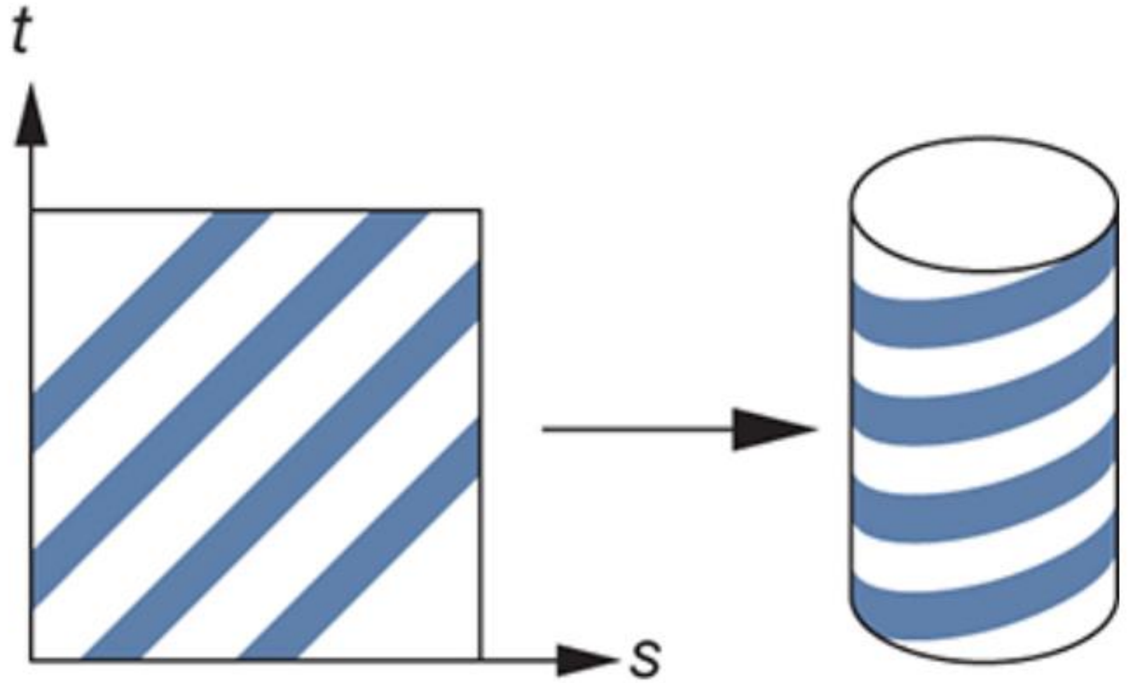
2D Texture Mapping

- Another approach to the mapping problem is to use a two-step mapping.
- The first step maps the texture to a simple three-dimensional intermediate surface, such as a sphere, cylinder, or cube.
- In the second step, the intermediate surface containing the mapped texture is mapped to the surface being rendered.
- This two-step mapping process can be applied to surfaces defined in either geometric or parametric coordinates.

2D Texture Mapping

- Suppose that our texture coordinates vary over a unit square and that we use the surface of a cylinder of height *h* and radius *r* as our intermediate object. Points on the cylinder are given by the parametric equations.

2D Texture Mapping



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

2D Texture Mapping

- *u* and *v* vary over (*0, 1*).
- We can use the mapping

$$s = u \quad t = v\,.$$

- By using only the curved part of the cylinder, and not the top and bottom, we are able to map the texture without distorting its shape.

2D Texture Mapping

- However, if we map to a closed object, such as a sphere, we must introduce shape distortion. This problem is similar to the problem of creating a two-dimensional image of the earth for a map. Both texture-mapping and map-design techniques must choose among a variety of representations, based on where we wish to place the distortion.

2D Texture Mapping

- For example, the familiar Mercator projection puts the most distortion at the poles. If we use a sphere of radius *r* as the intermediate surface, a possible mapping is

$$\begin{aligned} x &= r\cos(2\pi\, u) \\ y &= r\sin(2\pi\, \mathrm{u})\cos(2\pi\, u) \\ z &= r\sin(2\pi\, \mathrm{u})\sin(2\pi\, \mathrm{v}). \end{aligned}$$

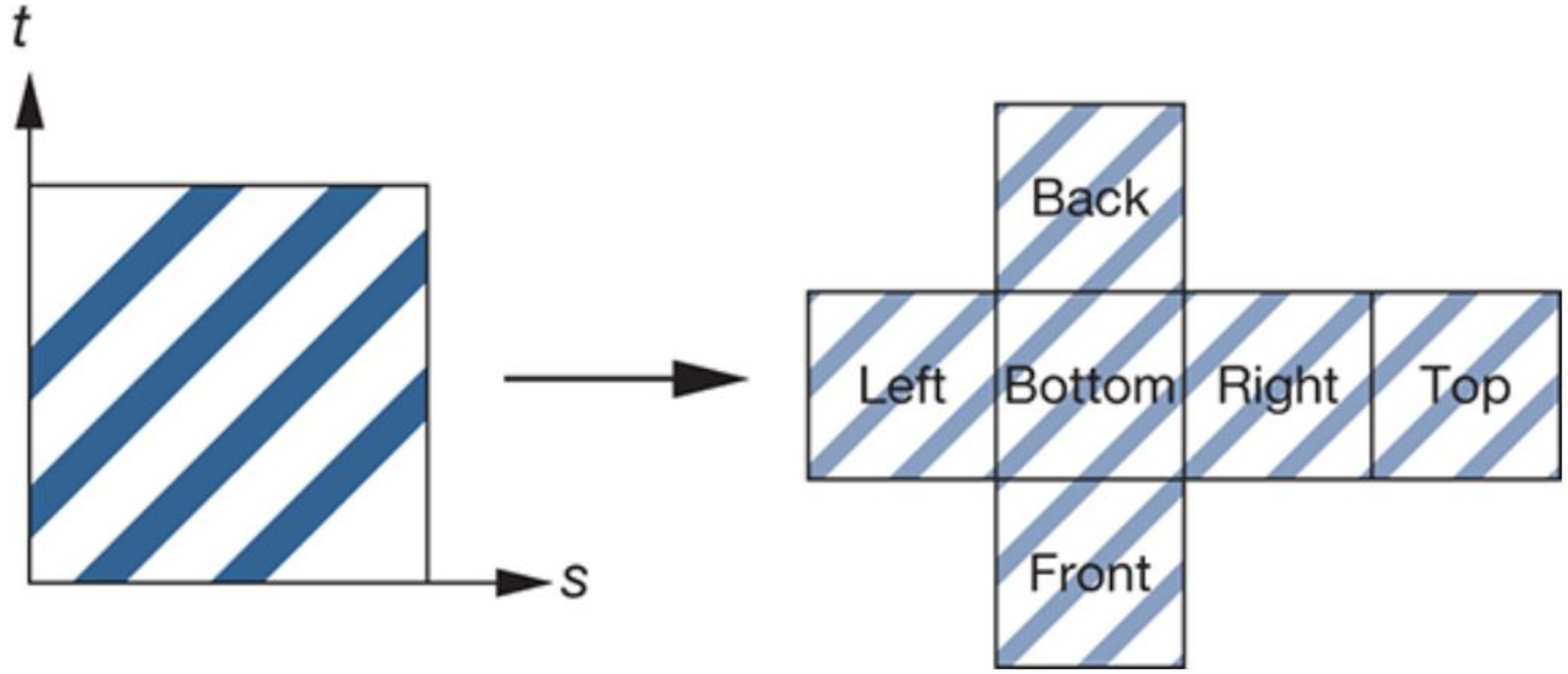
but is subject to distortion that is most pronounced at the poles.

2D Texture Mapping

- We can also use a rectangular box. Here, we map the texture to a box that can be unraveled, like a cardboard packing box. This mapping often is used with environment maps.

2D Texture Mapping

Figure 7.10 Texture mapping with a box.



2D Texture Mapping

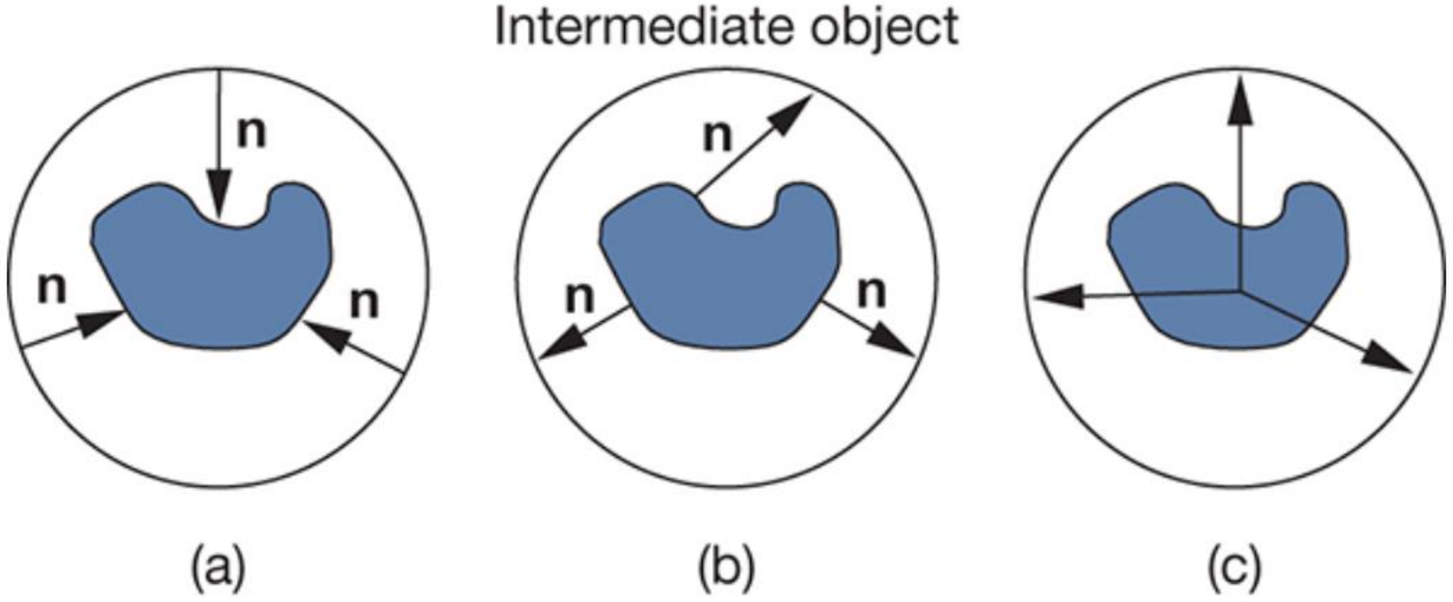
- The second step is to map the texture values on the intermediate object to the desired surface.
- Three possible strategies:
  - In panel (a), we take the texture value at a point on the intermediate object, go from this point in the direction of the normal until we intersect the object, and then place the texture value at the point of intersection.

2D Texture Mapping

- We could also reverse this method, starting at a point on the surface of the object and going in the direction of the normal at this point until we intersect the intermediate object, where we obtain the texture value, as shown in panel (b).
- A third option, if we know the center of the object, is to draw a line from the center through a point on the object and to calculate the intersection of this line with the intermediate surface, as shown in panel (c). The texture at the point of intersection with the intermediate object is assigned to the corresponding point on the desired object.

2D Texture Mapping

Figure 7.11 Second mapping. (a) Using the normal from the intermediate surface. (b) Using the normal from the object surface. (c) Using the center of the object.



Texture Mapping in WebGL

- Texture mapping requires interaction among the application program, the vertex shader, and the fragment shader. There are three basic steps. First, we must form a texture image and place it in texture memory on the GPU. Second, we must assign texture coordinates to each fragment. Finally, we must apply the texture to each fragment. Each of these steps can be accomplished in multiple ways, and there are many parameters that we can use to control the process.

Sample Program (*TexturedQuad.js*)

- This program is structured into five main parts:
  - 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 (initTextures()).
  - Part 5: Configure the loaded texture so that it can be used in WebGL (loadTexture()).

Sample Program (*TexturedQuad.js*)

```
// TexturedQuad.js
// Vertex shader program                                   <- (Part1)
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'attribute vec2 a_TexCoord;\n' +
  'varying vec2 v_TexCoord;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  v_TexCoord = a_TexCoord;\n' +
  '}\n';
```

Sample Program (*TexturedQuad.js*)

```
// Fragment shader program                            <- (Part2)
var FSHADER_SOURCE =
...
  'uniform sampler2D u_Sampler;\n' +
  'varying vec2 v_TexCoord;\n' +
  'void main() {\n' +
  '  gl_FragColor = texture2D(u_Sampler, v_TexCoord);\n' +
  '}\n';

```

Using Texture Coordinates (*initVertexBuffers()*)

```
function main() {
   ...
  // Set the vertices information                       <- (Part3)
  var n = initVertexBuffers(gl);
 ...
  // Setting the textures
  if (!initTextures(gl, n)) {
 ...
  }
}

function initVertexBuffers(gl) {
  var verticesTexCoords = new Float32Array([
    // Vertices coordinates, textures coordinates
    -0.5,  0.5,   0.0, 1.0,
    -0.5, -0.5,   0.0, 0.0,
     0.5,  0.5,   1.0, 1.0,
     0.5, -0.5,   1.0, 0.0,
  ]);
  var n = 4; // The number of vertices

  // Create the buffer object
  var vertexTexCoordBuffer = gl.createBuffer();
 ...
  // Write the vertex coords and textures coords to the object buffer
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexTexCoordBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, verticesTexCoords, gl.STATIC_DRAW);

  var FSIZE = verticesTexCoords.BYTES_PER_ELEMENT;
 ...
 gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE * 4, 0);
 gl.enableVertexAttribArray(a_Position);   // Enable buffer allocation

  // Allocate the texture coordinates to a_TexCoord, and enable it.
  var a_TexCoord = gl.getAttribLocation(gl.program, 'a_TexCoord');
 ...
  gl.vertexAttribPointer(a_TexCoord, 2, gl.FLOAT, false, FSIZE * 4, FSIZE * 2);
  gl.enableVertexAttribArray(a_TexCoord);  // Enable buffer allocation
 ...
  return n;
}
```

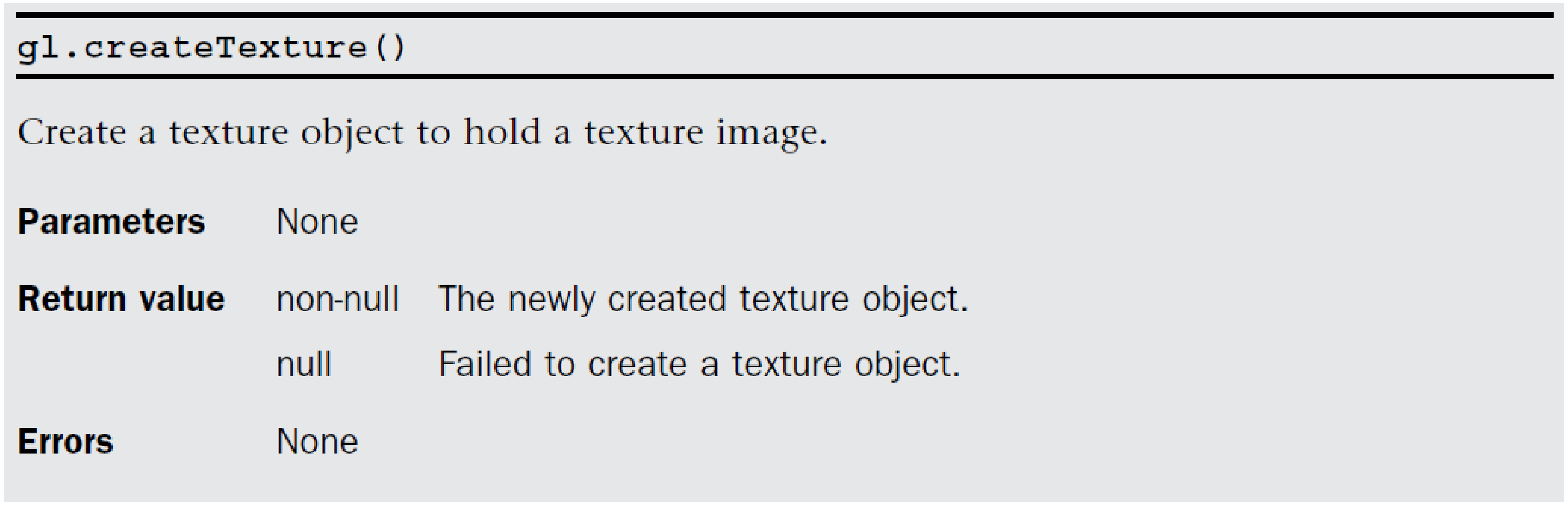
Setting Up and Loading Images (*initTextures()*)

```
function initTextures(gl, n)                                       <-  (Part4)
  var texture = gl.createTexture();   // Create a texture object
 ...
  // Get the storage location of the u_Sampler
  var u_Sampler = gl.getUniformLocation(gl.program, 'u_Sampler');
 ...
  var image = new Image();  // Create an image object
 ...
  // Register the event handler to be called on loading an image
  image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
  // Tell the browser to load an image
  image.src = '../resources/sky.jpg';

  return true;
}
```

Texture Object

- A texture object is used for managing the texture image in the WebGL system.
- A texture object is created using *gl.createTexture()*.



`gl.createTexture()`

Create a texture object to hold a texture image.

Parameters	None	
Return value	non-null	The newly created texture object.
	null	Failed to create a texture object.
Errors	None	

Texture Object

- This call creates the texture object in the WebGL system.
- *gl.TEXTURE0* to *gl.TEXTURE7* are texture units for managing a texture image, and each has an associated *gl.TEXTURE_2D*, which is the texture target for specifying the type of texture.

Texture Object

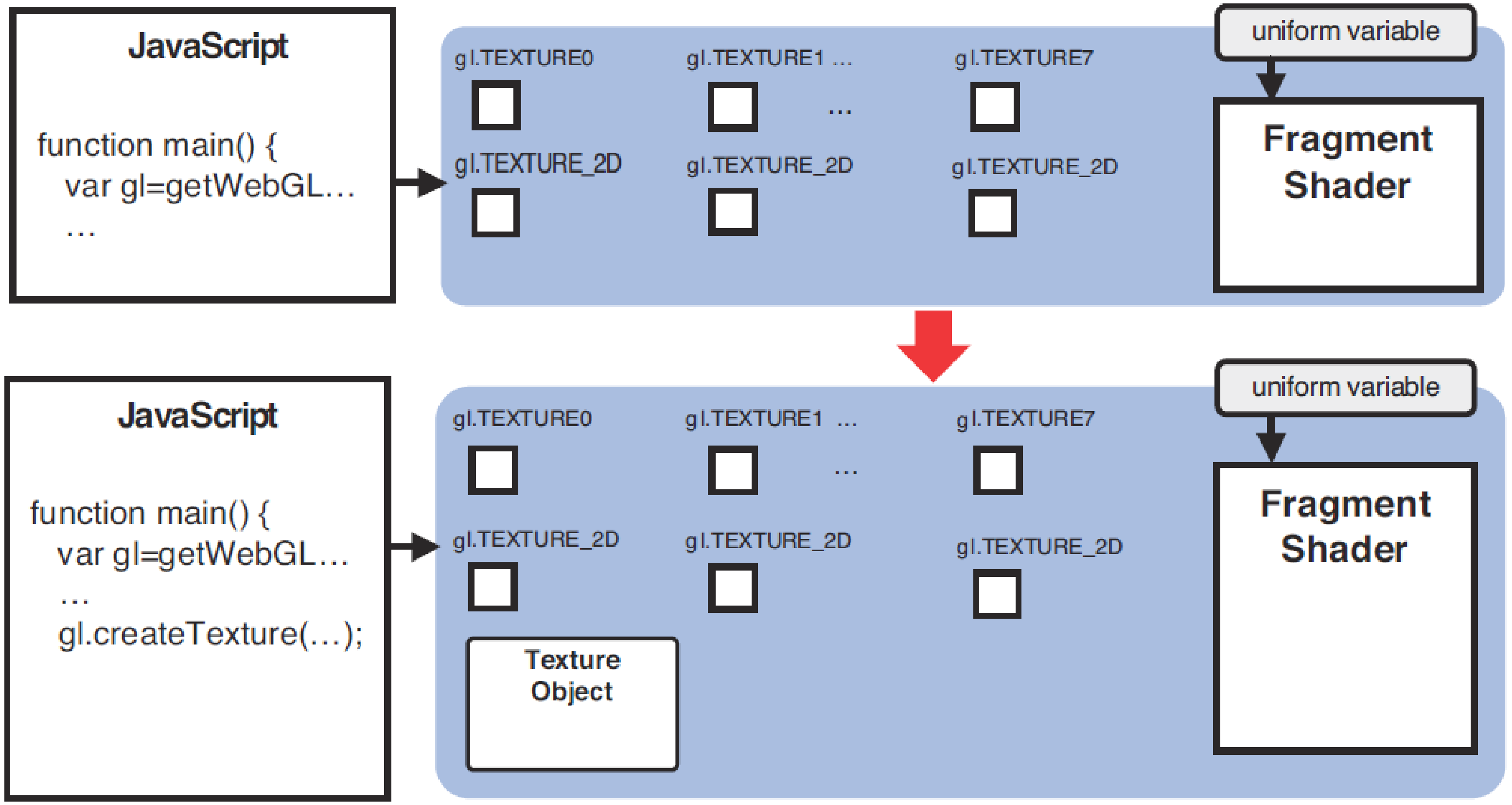
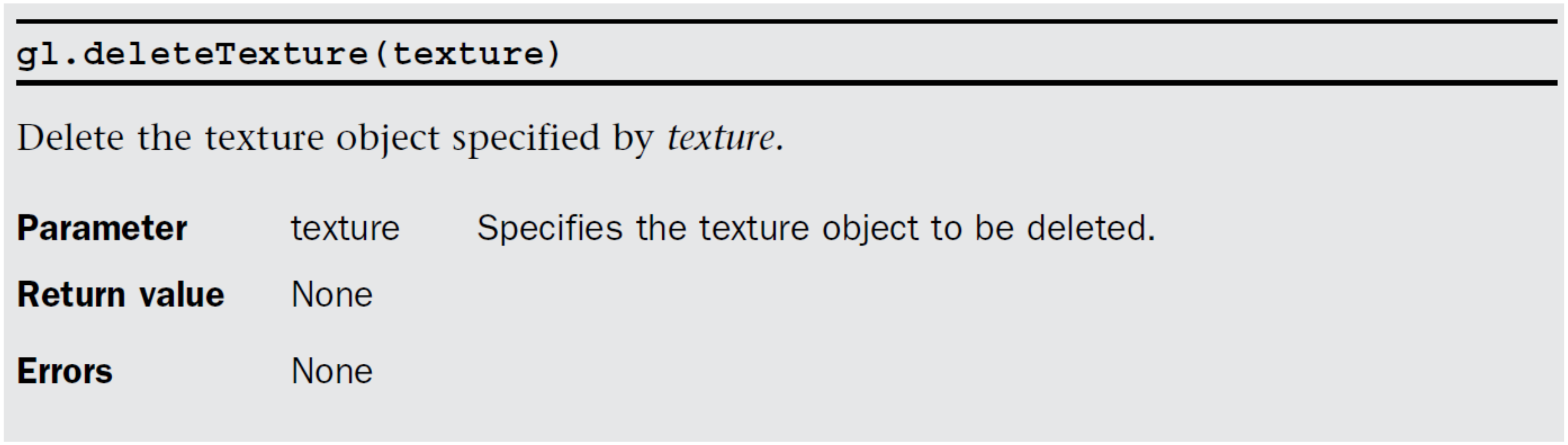


Figure 5.22 Create a texture object

Texture Object

- The texture object can be deleted using *gl.deleteTexture()*. Note, if this method is called with a texture object that has already been deleted, the call has no effect.



gl.deleteTexture(texture)

Delete the texture object specified by *texture*.

Parameter	texture	Specifies the texture object to be deleted.
Return value	None	
Errors	None	

Image Object

- Next, we need to use an *Image* object to request that the browser load the image that will be mapped to the rectangle.

```
var image = new Image();  // Create an Image object
...
// Register an event handler to be called when image loading completes
image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
// Tell the browser to load an image
image.src = '../resources/sky.jpg';
```

- This code snippet creates an *Image* object, registers the event handler (*loadTexture()*) to be called on loading the image, and tells the browser to load the image.

Setting Up and Loading Images (*initTextures()*)

- Because loading of images is performed asynchronously, when the browser signals completion of loading, it needs to pass the image to the WebGL system. Line *120* handles this, telling the browser that, after loading the image, the anonymous function *loadTexture()* should be called.

```
120   image.onload = function(){ loadTexture(gl, n, texture, u_Sampler, image); };
```

Setting Up and Loading Images (*initTextures()*)

- *loadTexture()* takes five parameters, with the newly loaded image being passed via the variable (*Image* object) as the last argument *image*. *gl* is the rendering context for WebGL, *n* is the number of vertices, *texture* is the texture object created, and *u_Sampler* is the storage location of a uniform variable.

Asynchronous Loading Texture Images

- Usually, OpenGL applications written in C or C++ load the texture image files straight from the hard disk where they are stored.
- However, because WebGL programs are running inside the browser, it is impossible to load images directly. Instead, it is necessary to read images indirectly by requesting the browser to do it. (Typically, the browser sends a request to the web server to obtain the image.)

Asynchronous Loading Texture Images

- The advantage is that you can use any kind of image a browser can display, but it also makes the process more complex because you now have to handle two processes (the browser loading request, and the actual WebGL loading) that behave “asynchronously” (they run in the background) and thus do not block execution of the program.

Make the Texture Ready to Use in the WebGL System (*loadTexture()*)

```
function loadTexture(gl, n, texture, u_Sampler, image){   <- (Part5)
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1); // Flip the image's y axis
  // Enable the texture unit 0
  gl.activeTexture(gl.TEXTURE0);
  // Bind the texture object to the target
  gl.bindTexture(gl.TEXTURE_2D, texture);

  // Set the texture parameters
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  // Set the texture image
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);

  // Set the texture unit 0 to the sampler
  gl.uniform1i(u_Sampler, 0);
...
  gl.drawArrays(gl.TRIANGLE_STRIP, 0, n); // Draw a rectangle
}
```

Flip an Image’s Y-Axis

```
128   gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, 1);// Flip the image's y-axis
```

- The *t*-axis direction of the WebGL texture coordinate system is the inverse of the *y*-axis direction of the coordinate system used by PNG, BMP, JPG, and so on. For this reason, if you flip the image’s Y-axis, you can map the image to the shape correctly.

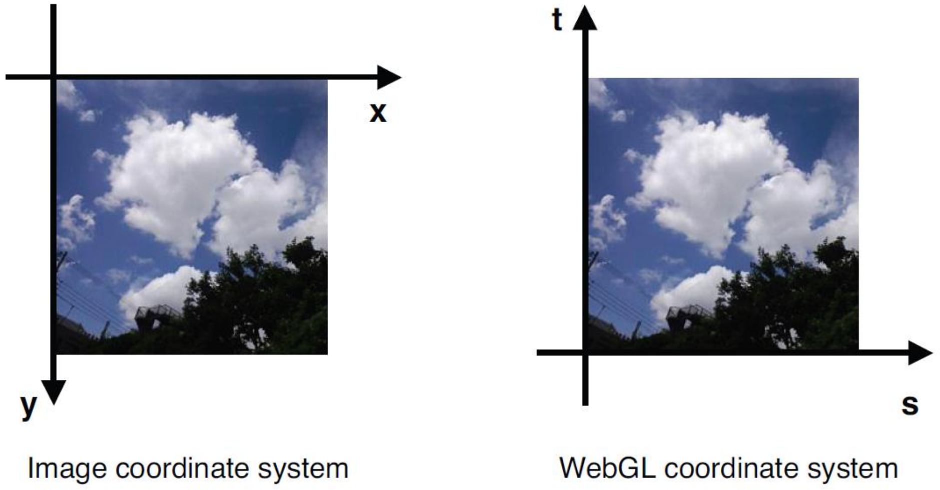
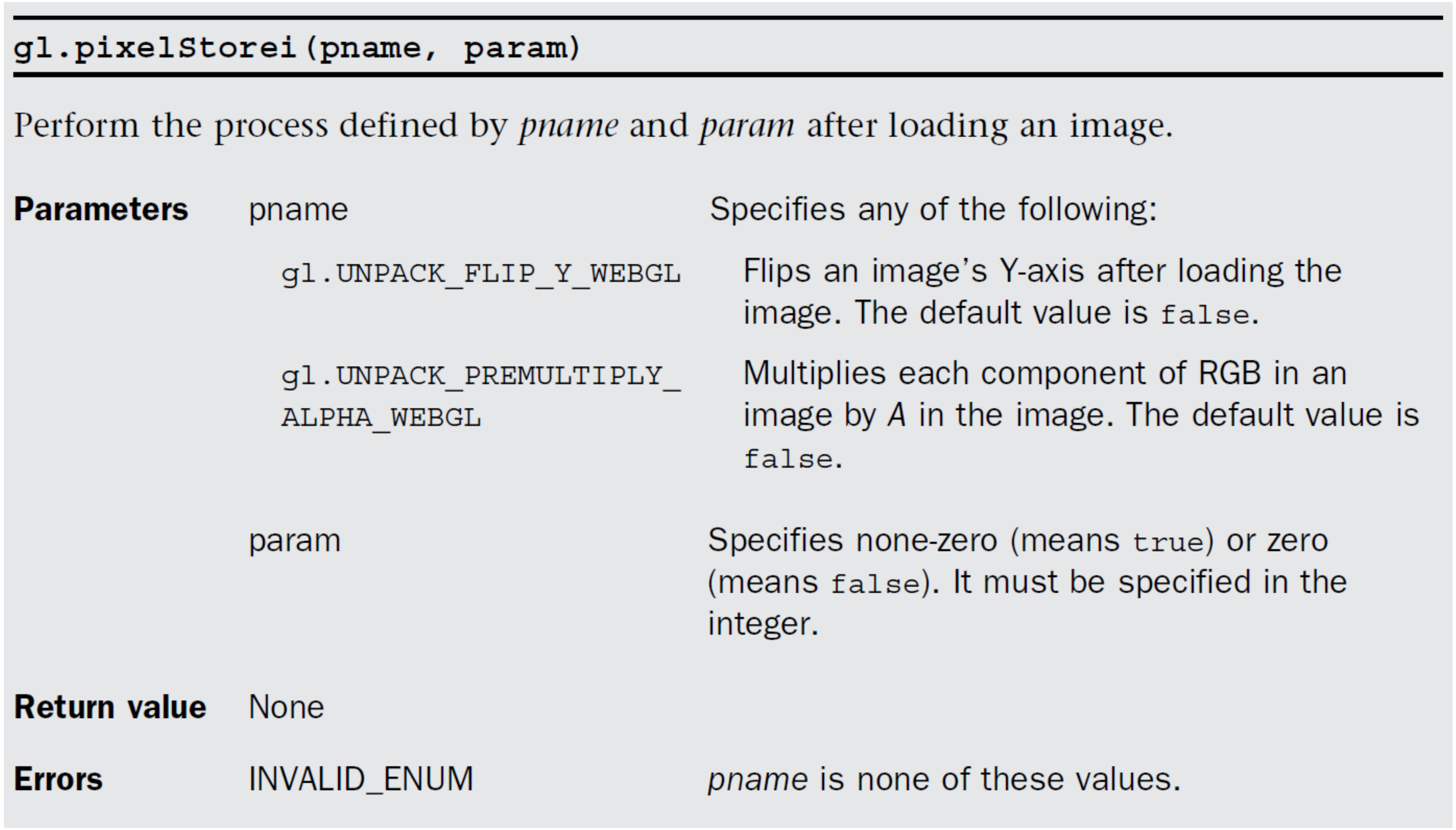


Figure 5.24 The image coordinate system and WebGL texture coordinate system

Flip an Image’s Y-Axis



`gl.pixelStorei(pname, param)`

Perform the process defined by *pname* and *param* after loading an image.

Parameters	pname	Specifies any of the following:
	`gl.UNPACK_FLIP_Y_WEBGL`	Flips an image’s Y-axis after loading the image. The default value is `false`.
	`gl.UNPACK_PREMULTIPLY_ALPHA_WEBGL`	Multiplies each component of RGB in an image by *A* in the image. The default value is `false`.
	param	Specifies none-zero (means `true`) or zero (means `false`). It must be specified in the integer.
Return value	None	
Errors	INVALID_ENUM	*pname* is none of these values.

Making a Texture Unit Active (*gl.activeTexture()*)

- WebGL supports multiple texture images (multitexturing) using a mechanism called a *texture unit*.
- A texture unit manages texture images by using a unit number for each texture.
- Because of this, even if you only want to use a single texture image, you must specify and use a texture unit.

Making a Texture Unit Active (*gl.activeTexture()*)

```
// Make the texture unit 0 active
gl.activeTexture(gl.TEXTURE0);
```

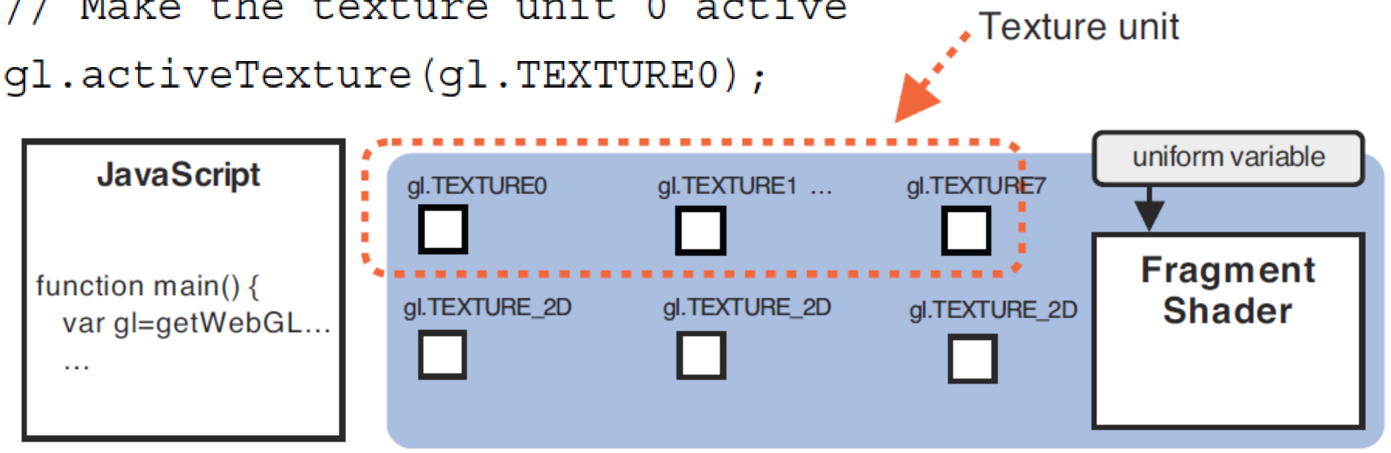


Figure 5.25 Multiple texture units managed by WebGL

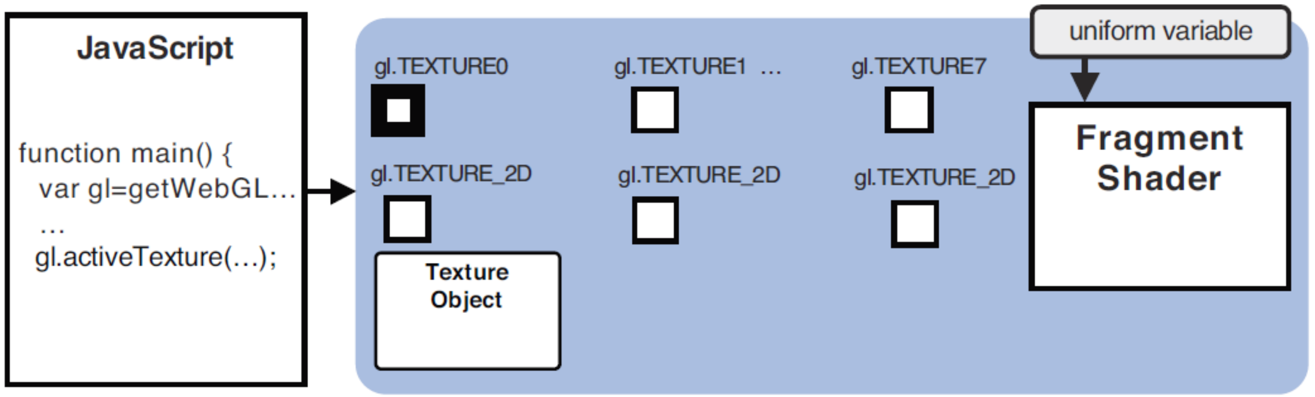


Figure 5.26 Activate texture unit (gl.TEXTURE0)

Binding a Texture Object to a Target (*gl.bindTexture()*)

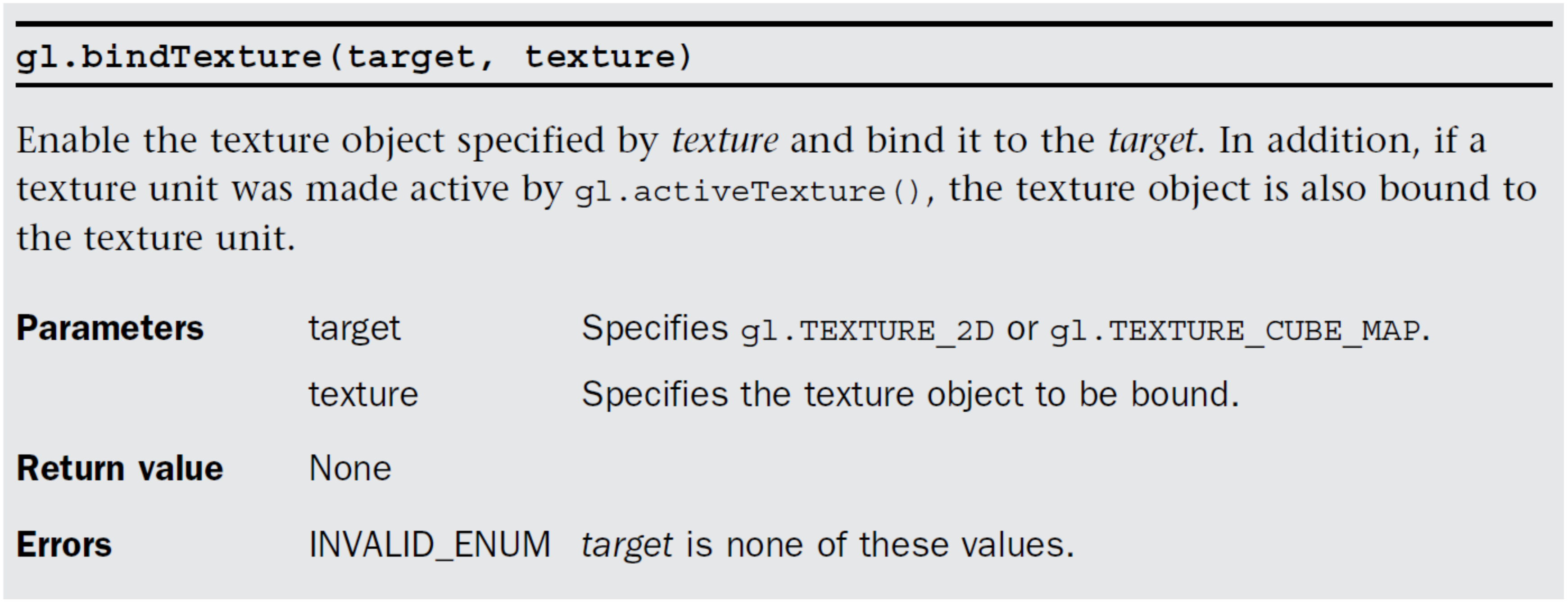
- Next, we need to tell the WebGL system what types of texture image is used in the texture object.

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

Table 5.2 Types of Textures

Type of Texture	Description
`gl.TEXTURE_2D`	Two-dimensional texture
`gl.TEXTURE_CUBE_MAP`	Cube map texture

Binding a Texture Object to a Target (*gl.bindTexture()*)



`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.

Parameters	target	Specifies `gl.TEXTURE_2D` or `gl.TEXTURE_CUBE_MAP`.
	texture	Specifies the texture object to be bound.
Return value	None	
Errors	INVALID_ENUM	*target* is none of these values.

This method performs two tasks: enabling the texture object and binding it to target, and binding it to the texture unit. In this case, because the texture unit 0 (gl.TEXTURE0) is active, after executing, the internal state of the WebGL system is changed.

Binding a Texture Object to a Target (*gl.bindTexture()*)

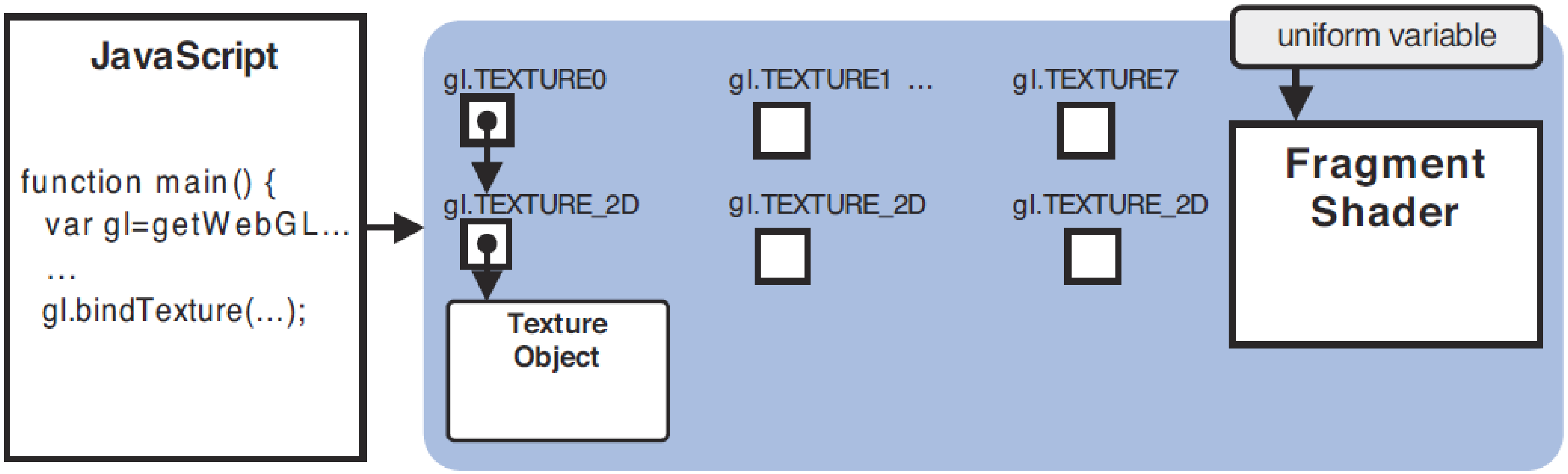


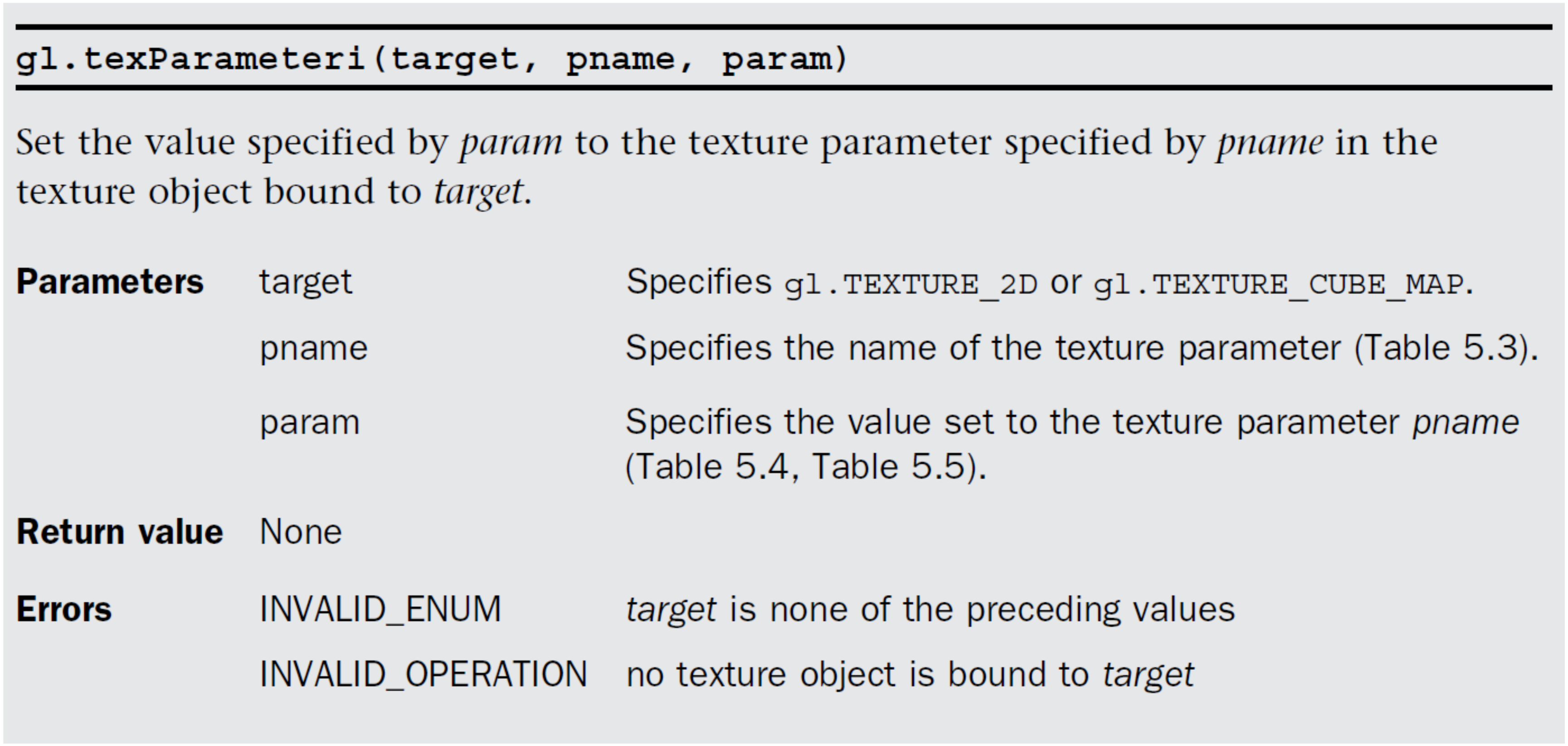
Figure 5.27 Bind a texture object to the target

Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)

- In the next step, we need to set the parameters (texture parameter) that specify how the texture image will be processed when the texture image is mapped to shapes.
- The function *gl.texParameteri()* can be used to set texture parameters.

```
// Set the texture parameters
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
```

Set the Texture Parameters of a Texture Object (*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*

Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)

Table 5.3 Texture Parameters and Their Default Values

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`

Table 5.4 Non-Mipmapped Values, Which Can be Specified to gl.TEXTURE_MAG_FILTER and gl.TEXTURE_MIN_FILTER[3]

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.)

Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)

Table 5.5 Values that Can be Specified to gl.TEXTURE_WRAP_S and gl.TEXTURE_WRAP_T

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

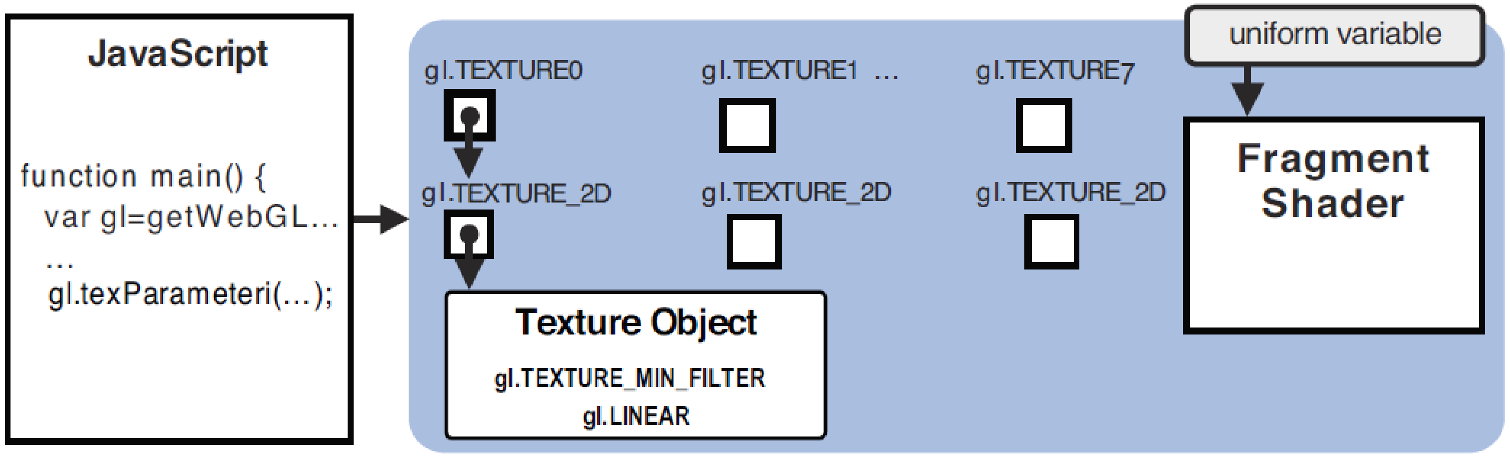
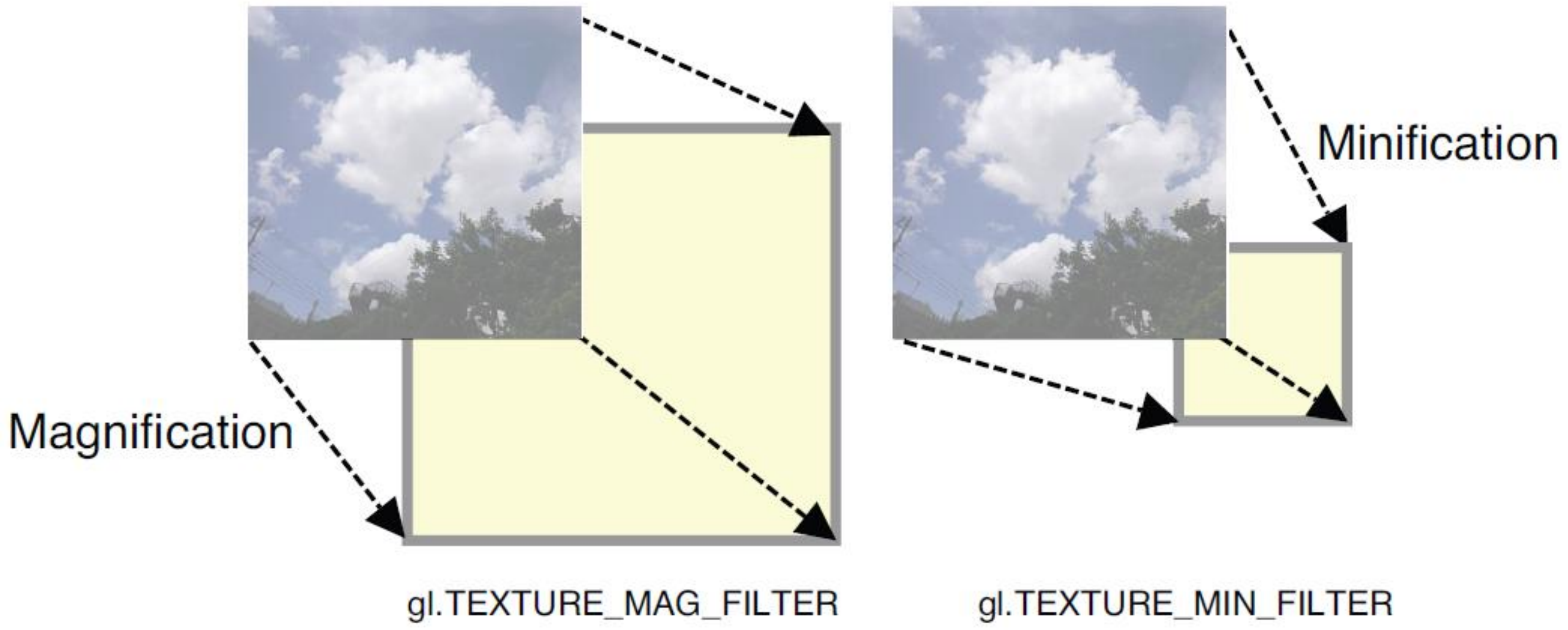


Figure 5.29 Set texture parameter

Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)

- 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.

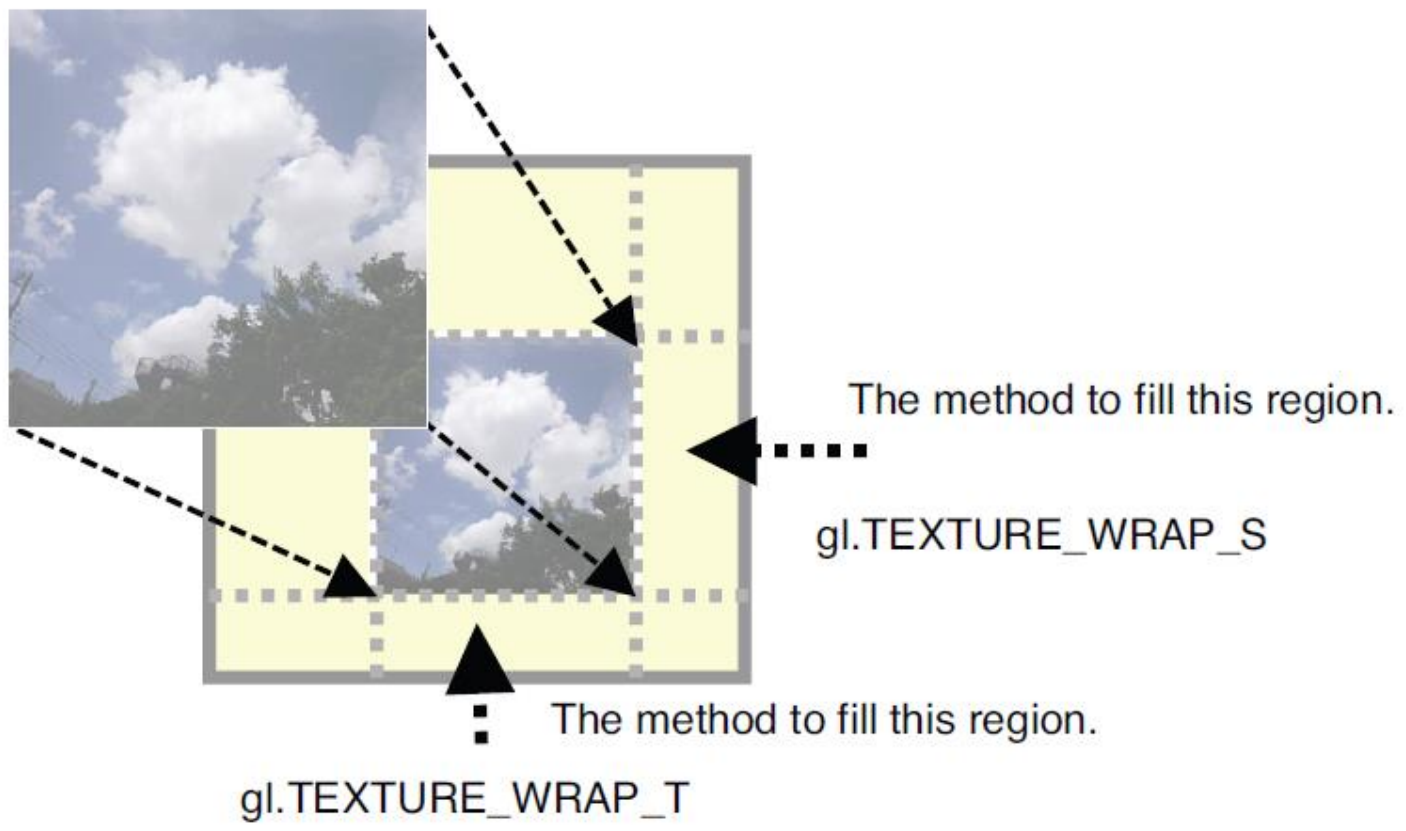
Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)



Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)

- 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): the method used to fill the remaining regions in the top and bottom of a subregion.

Set the Texture Parameters of a Texture Object (*gl.texParameteri()*)

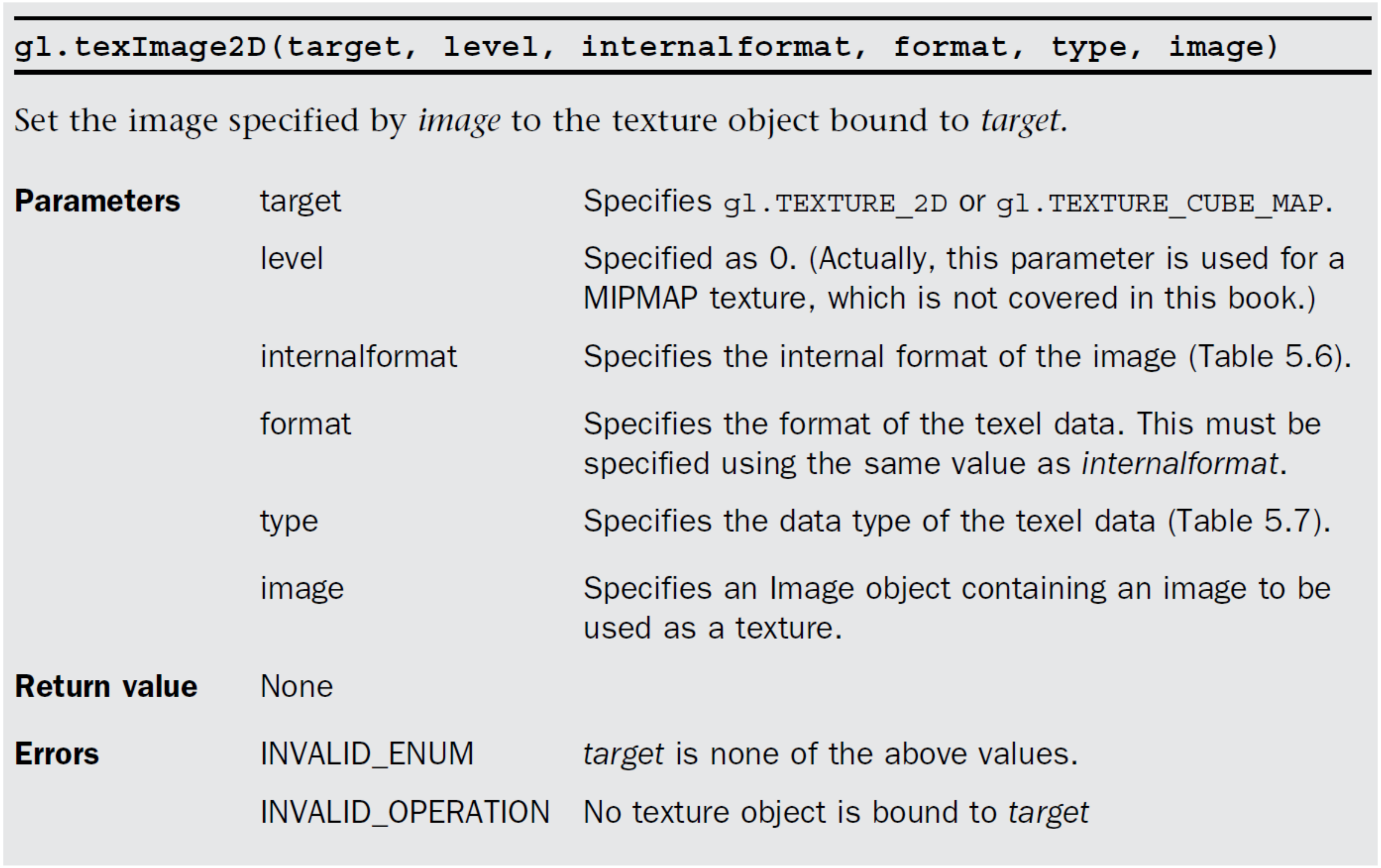

The method to fill this region.
gl.TEXTURE_WRAP_S
The method to fill this region.
gl.TEXTURE_WRAP_T

Assigning a Texture Image to a Texture Object (*gl.texImage2D()*)

- To assign an image to a texture object, you use the method *gl.texImage2D().*
- In addition to assigning a texture, this method allows you to tell the WebGL system about the image characteristics.

```
// Set the texture image
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
```

Assigning a Texture Image to a Texture Object (*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*

Assigning a Texture Image to a Texture Object (*gl.texImage2D()*)

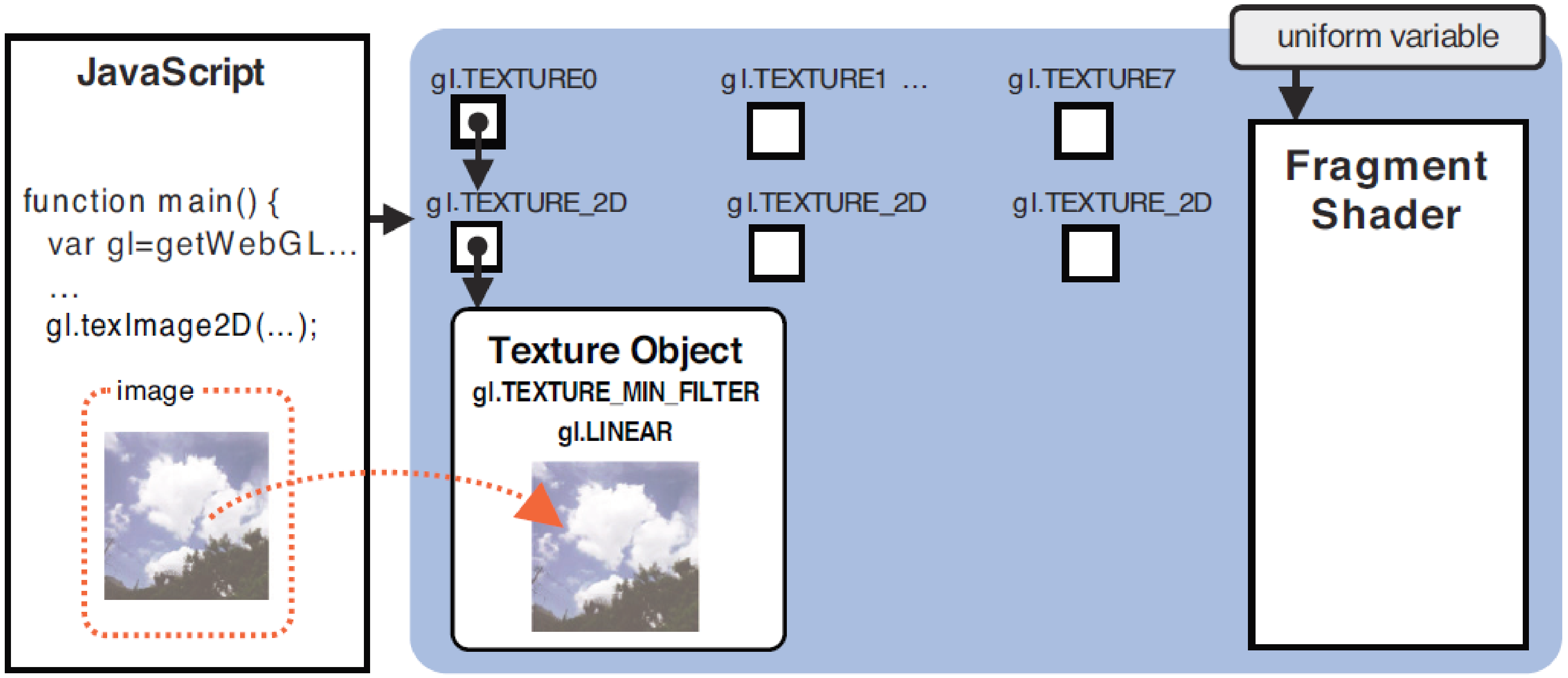


Figure 5.30 Assign an image to the texture object

Pass the Texture Unit to the Fragment Shader (*gl.uniform1i()*)

- Once the texture image has been passed to the WebGL system, it must be passed to the fragment shader to map it to the surface of the shape.

```
// Set the texture unit 0 to the sampler
gl.uniform1i(u_Sampler, 0);
```

Pass the Texture Unit to the Fragment Shader (*gl.uniform1i()*)

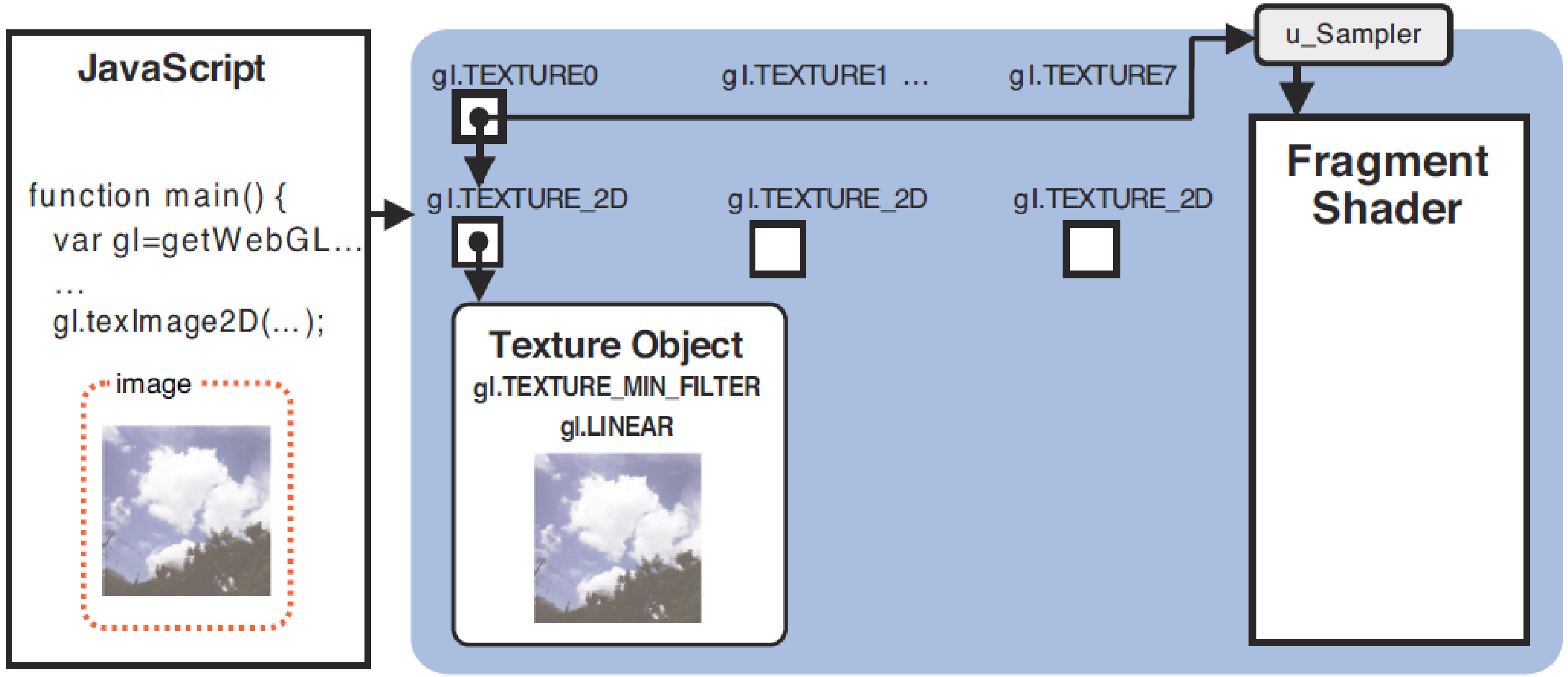
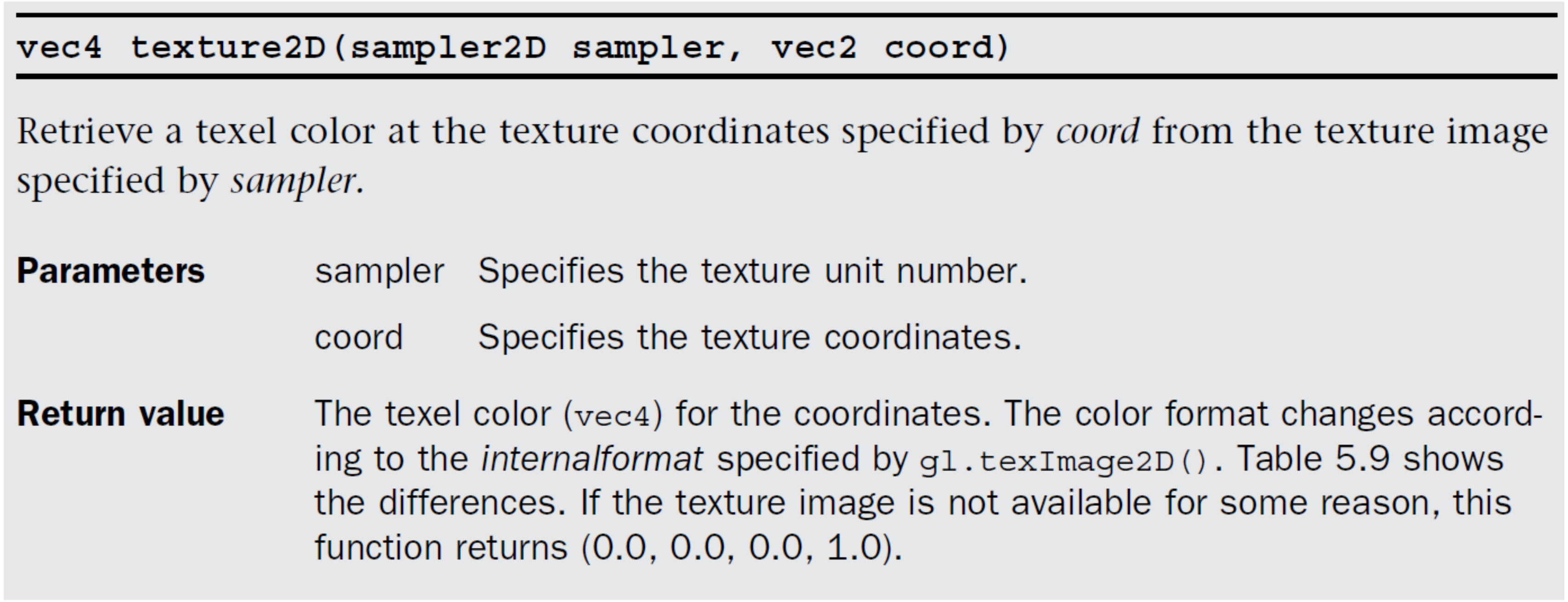


Figure 5.31 Set texture unit to uniform variable

Retrieve the Texel Color in a Fragment Shader (*texture2D()*)

' gl_FragColor = texture2D(u_Sampler, v_TexCoord);\n' +



`vec4 texture2D(sampler2D sampler, vec2 coord)`

Retrieve a texel color at the texture coordinates specified by *coord* from the texture image specified by *sampler*.

Parameters	sampler	Specifies the texture unit number.
	coord	Specifies the texture coordinates.
Return value	The texel color (`vec4`) for the coordinates. The color format changes according to the *internalformat* specified by `gl.texImage2D()`. Table 5.9 shows the differences. If the texture image is not available for some reason, this function returns (0.0, 0.0, 0.0, 1.0).	