CAPITULO 2
OBJETOS GEOMÉTRICOS Y TRANSFORMACIONES

CAPITULO 3
PROPIEDADES Y RENDERING

2.2 Transformaciones Geométricas en 2D

3.1 Color, Luz, materiales y textura

2.2.3 Textures

Textures – Texture Units

Why the sampler2D variable is a uniform if we didn't even assign it some value with ***glUniform?****.*

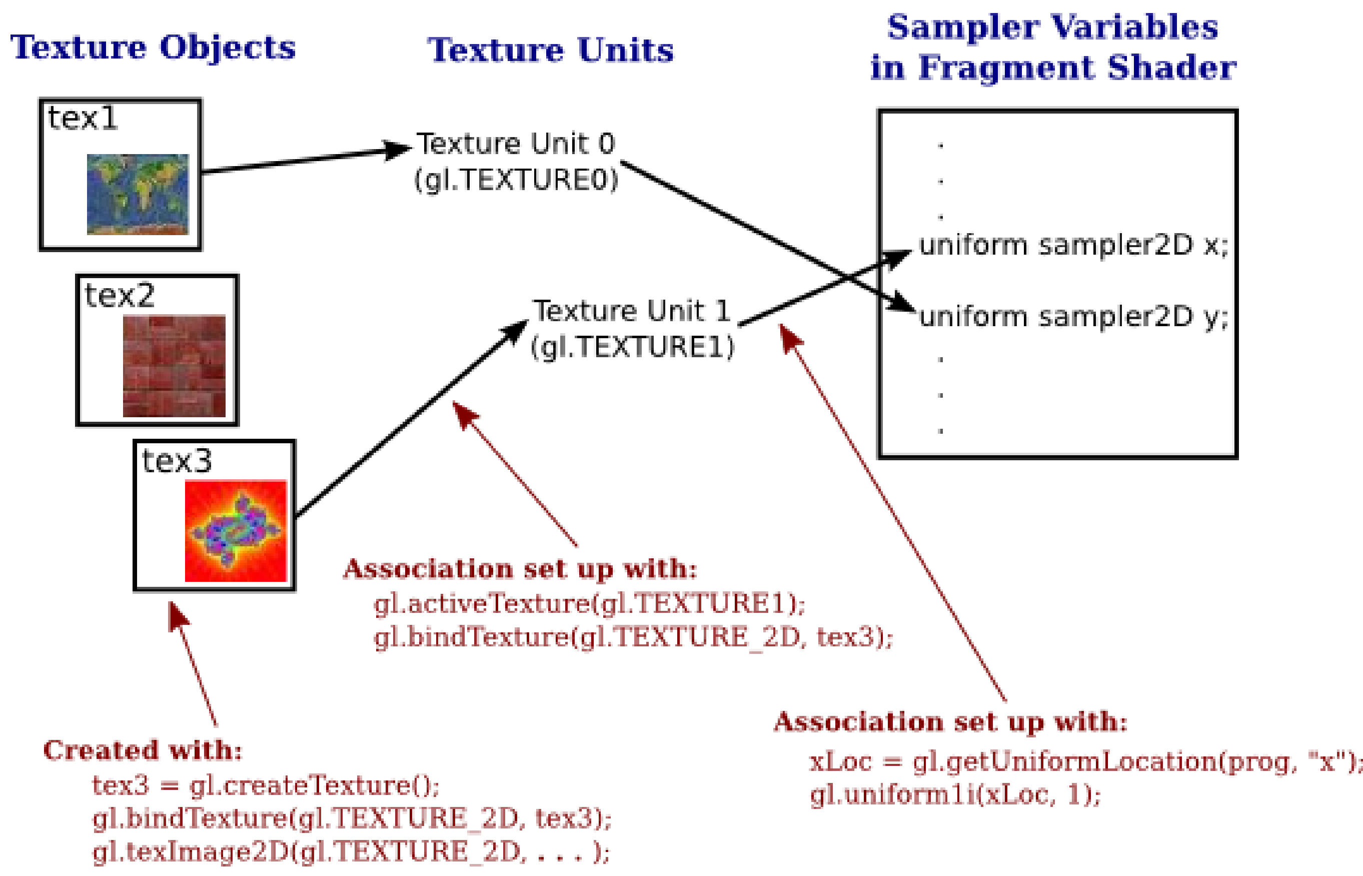
- Using **glUniform1i** we can actually assign a location value to the texture sampler so we can **set multiple textures** at once in a fragment shader.
- This location of a texture is more commonly known as a **texture unit.**

The default texture unit for a texture is 0 which is the default active texture unit so we didn't need to assign a location in the previous section; note that not all graphics drivers assign a default texture unit so the previous section might not've rendered for you.

Textures – Texture Units

- The main purpose of **texture units** is to allow us to use **more than 1 texture** in our shaders.
- By assigning texture units to the samplers, we can bind to multiple textures at once as long as we activate the corresponding texture unit first.
- Just like **glBindTexture** we can activate texture units using **glActiveTexture** passing in the texture unit we'd like to use:

```
// activate the texture unit first before binding texture
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
```



Texture Objects
Texture Units
Sampler Variables in Fragment Shader
tex1
tex2
tex3
Texture Unit 0 (gl.TEXTURE0)
Texture Unit 1 (gl.TEXTURE1)
uniform sampler2D x;
uniform sampler2D y;

Association set up with:
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);

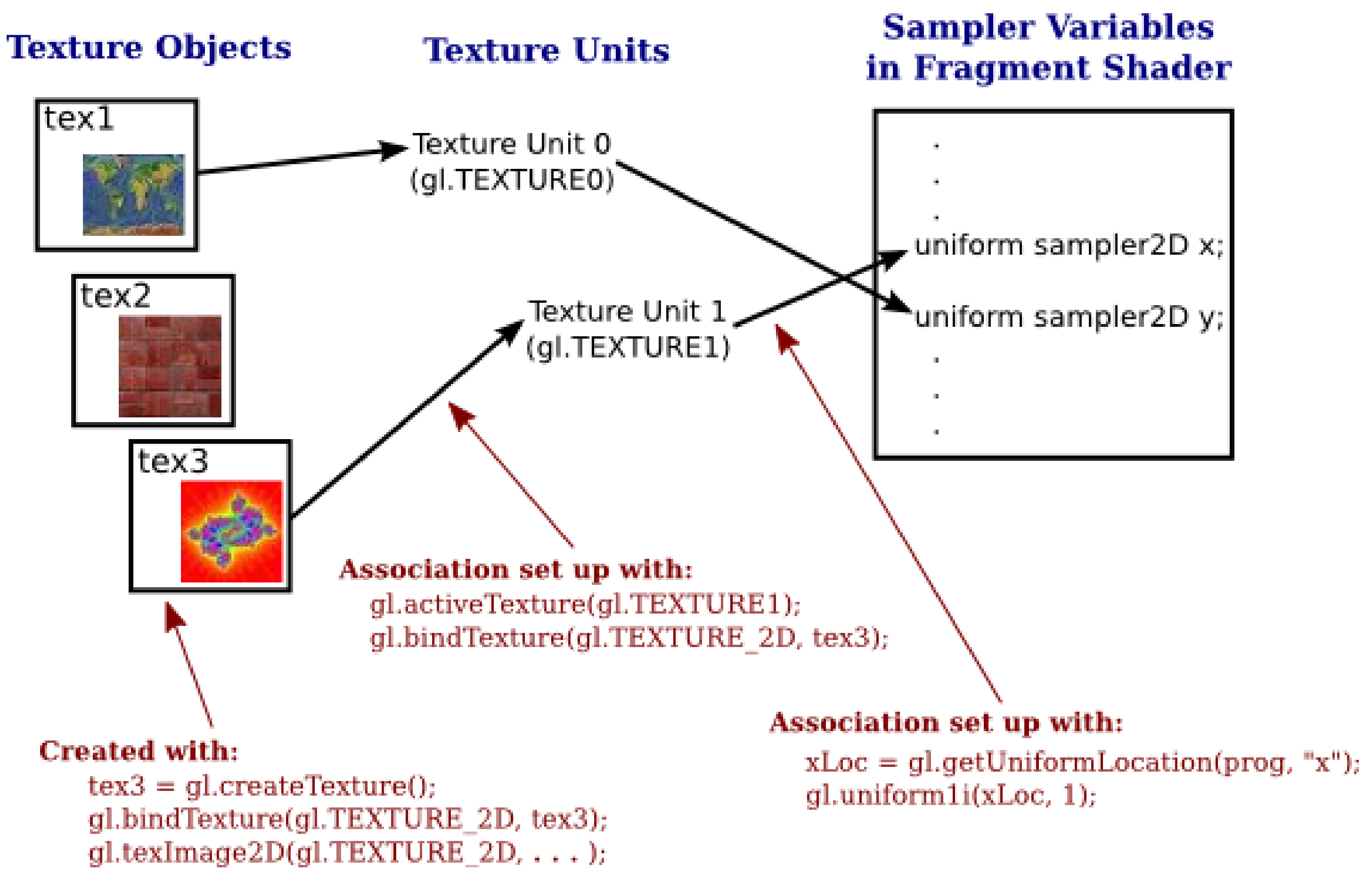
Created with:
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . .);

Association set up with:
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);

Textures – Texture Units

- After activating a texture unit, a subsequent **glBindTexture** call will bind that texture to the currently active texture unit.

- Texture unit **GL_TEXTURE0 is always by default** activated, so we didn't have to activate any texture units in the previous example when using glBindTexture.



Texture Objects
Texture Units
Sampler Variables in Fragment Shader
tex1
tex2
tex3
Texture Unit 0 (gl.TEXTURE0)
Texture Unit 1 (gl.TEXTURE1)
uniform sampler2D x;
uniform sampler2D y;

Created with:
```
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . . );
```

Association set up with:
```
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
```

Association set up with:
```
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);
```

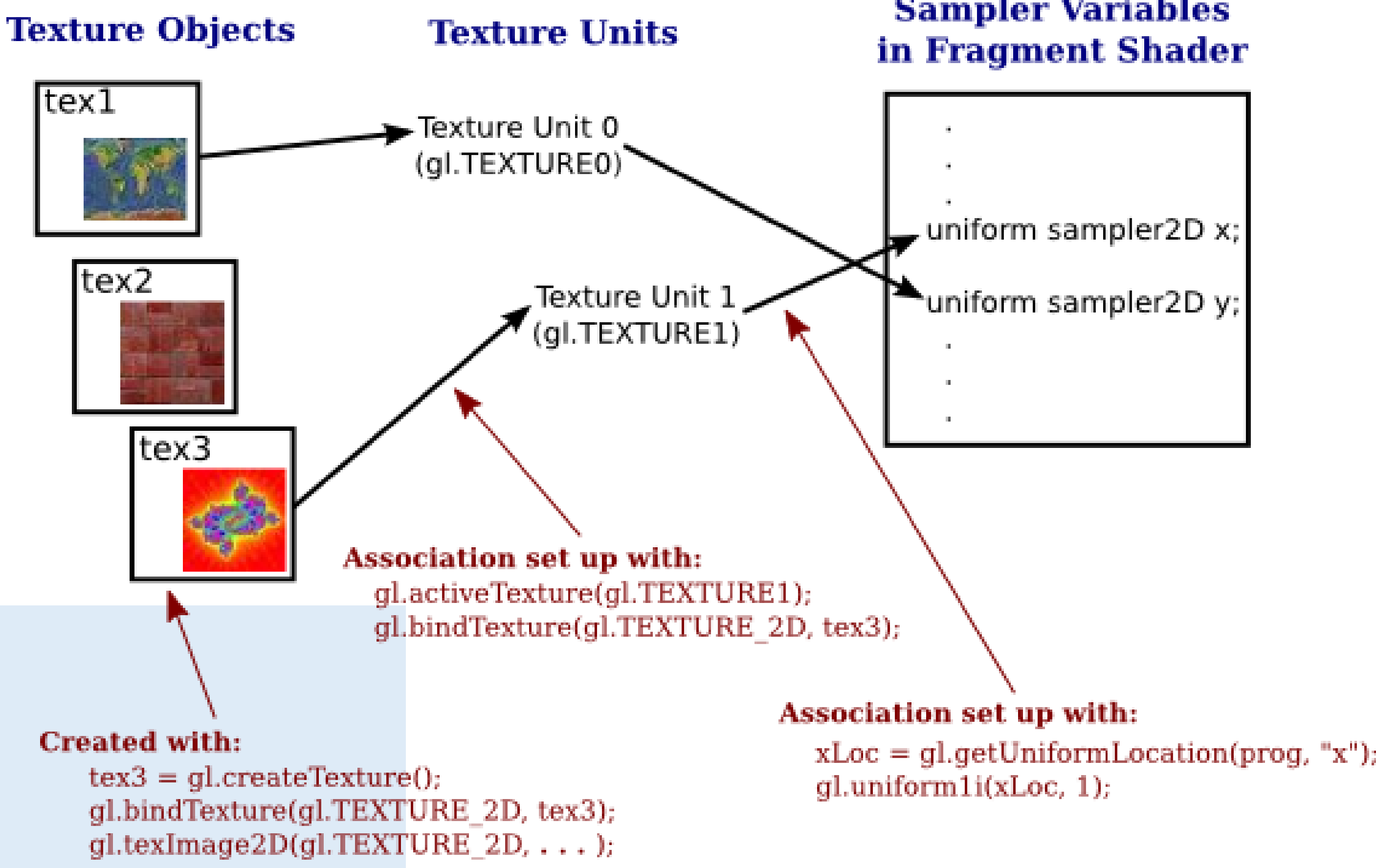
OpenGL should have a at least a minimum of **16 texture units for** you to use which you can activate using **GL_TEXTURE0 to GL_TEXTURE15**. They are defined in order so we could also get **GL_TEXTURE8** via **GL_TEXTURE0 + 8** for example, which is useful when we'd have to loop over several texture units.

Textures – Texture Units

We still however need to edit the **fragment shader** to accept another sampler.

```
#version 330 core
...
uniform sampler2D texture1;
uniform sampler2D texture2;

void main()
{
FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);
}
```



Association set up with:

```
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
```

Created with:

```
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . . );
```

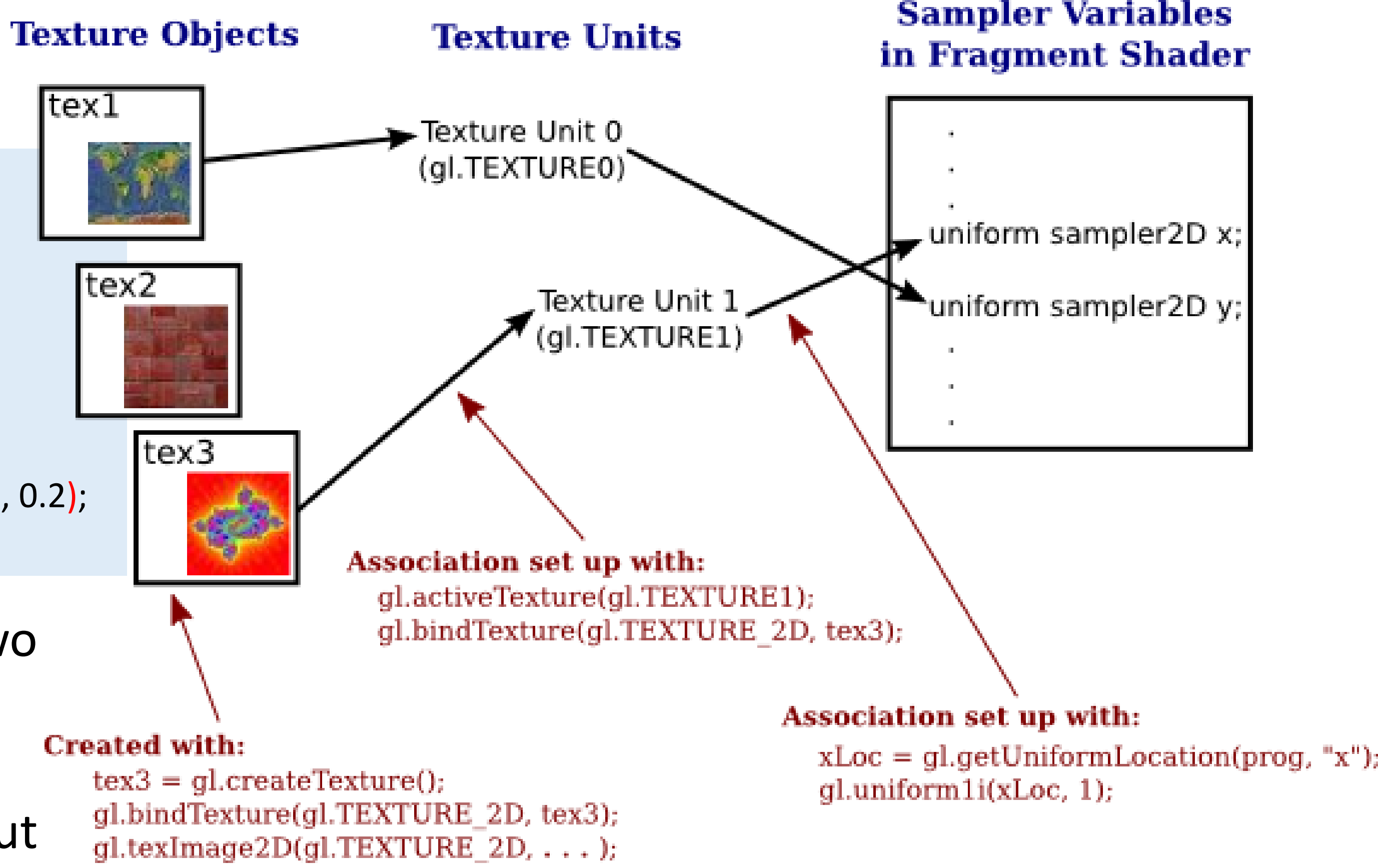
Association set up with:

```
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);
```

Textures – Texture Units

```
#version 330 core
...
uniform sampler2D texture1;
uniform sampler2D texture2;

void main()
{
FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);
}
```


Texture Objects
Texture Units
Sampler Variables in Fragment Shader
tex1
tex2
tex3
Texture Unit 0 (gl.TEXTURE0)
Texture Unit 1 (gl.TEXTURE1)
uniform sampler2D x;
uniform sampler2D y;

Association set up with:

```
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
```

Created with:

```
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . . );
```

Association set up with:

```
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);
```

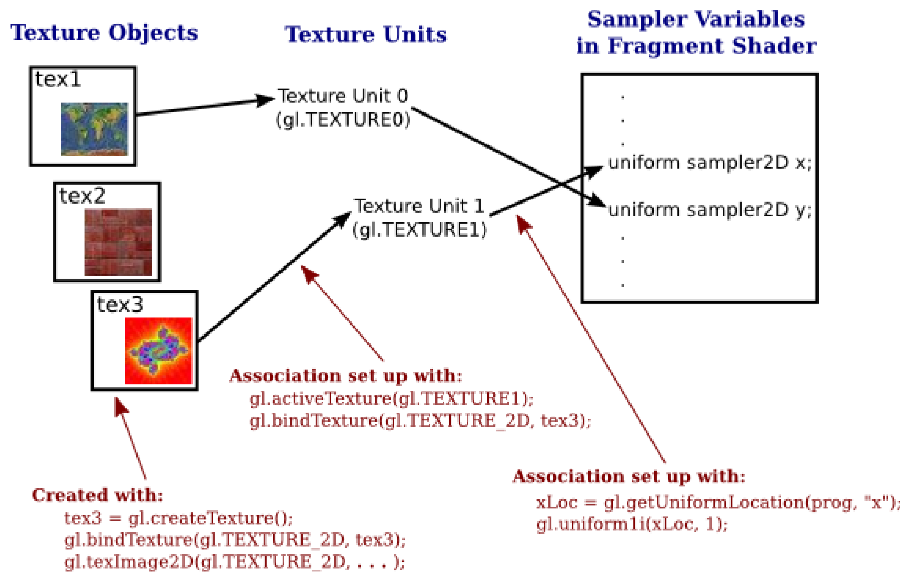
- The final output color is now the combination of two texture lookups.
- GLSL's built-in mix function takes two values as input and **linearly interpolates** between them based on its third argument.
- If the third value is 0.0 it returns the first input; if it's 1.0 it returns the second input value.
- A value of **0.2** will return **80% of the first** input color and **20% of the second** input color, resulting in a mixture of both our textures.

Textures – Texture Units

Create another texture object, load the image and generate the final texture using **glTexImage2D**.

```
unsigned char *data = stbi_load("awesomeface.png", &width, &height, &nrChannels, 0);
if (data)
{
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
   glGenerateMipmap(GL_TEXTURE_2D);
}
```

- Note that we now load a **.png image** that includes an alpha (transparency) channel.
- This means we now need to specify that the image data contains an alpha channel as well by using **GL_RGBA**; otherwise OpenGL will incorrectly interpret the image data.



```
. . .
uniform sampler2D x;
uniform sampler2D y;
. . .
```

Association set up with:

```
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
```

Created with:

```
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . . );
```

Association set up with:

```
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);
```

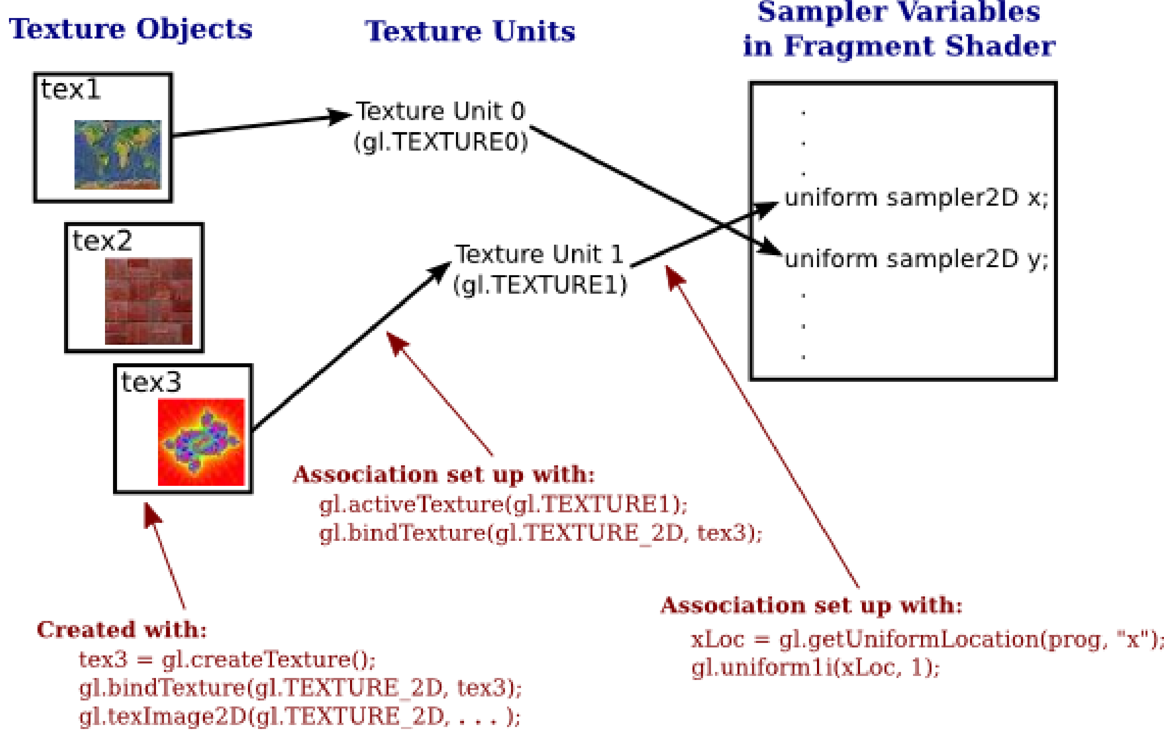
Image source: https://learnopengl.com/img/textures/awesomeface.png

Textures – Texture Units

To use the second texture (and the first texture) we'd have to change the rendering procedure a bit by binding both textures to the corresponding texture unit:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture2);

glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```



```
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
```

```
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . . );
```

```
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);
```

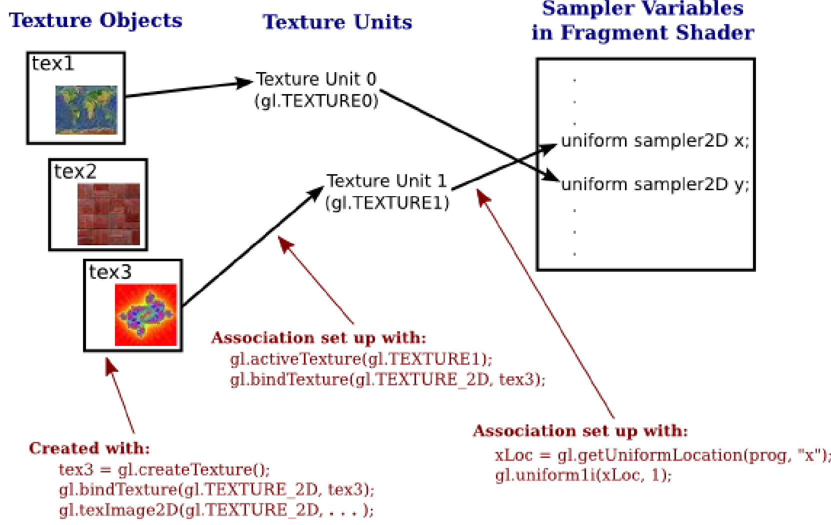
- We also have to tell OpenGL to which texture unit each shader sampler belongs to by setting each sampler using glUniform1i. We only have to set this once, so we can do this before we enter the render loop:

```
ourShader.use(); // don't forget to activate the shader before setting uniforms!
glUniform1i(glGetUniformLocation(ourShader.ID, "texture1"), 0); // set it manually
ourShader.setInt("texture2", 1); // or with shader class

while(...)
{
   [...]
}
```

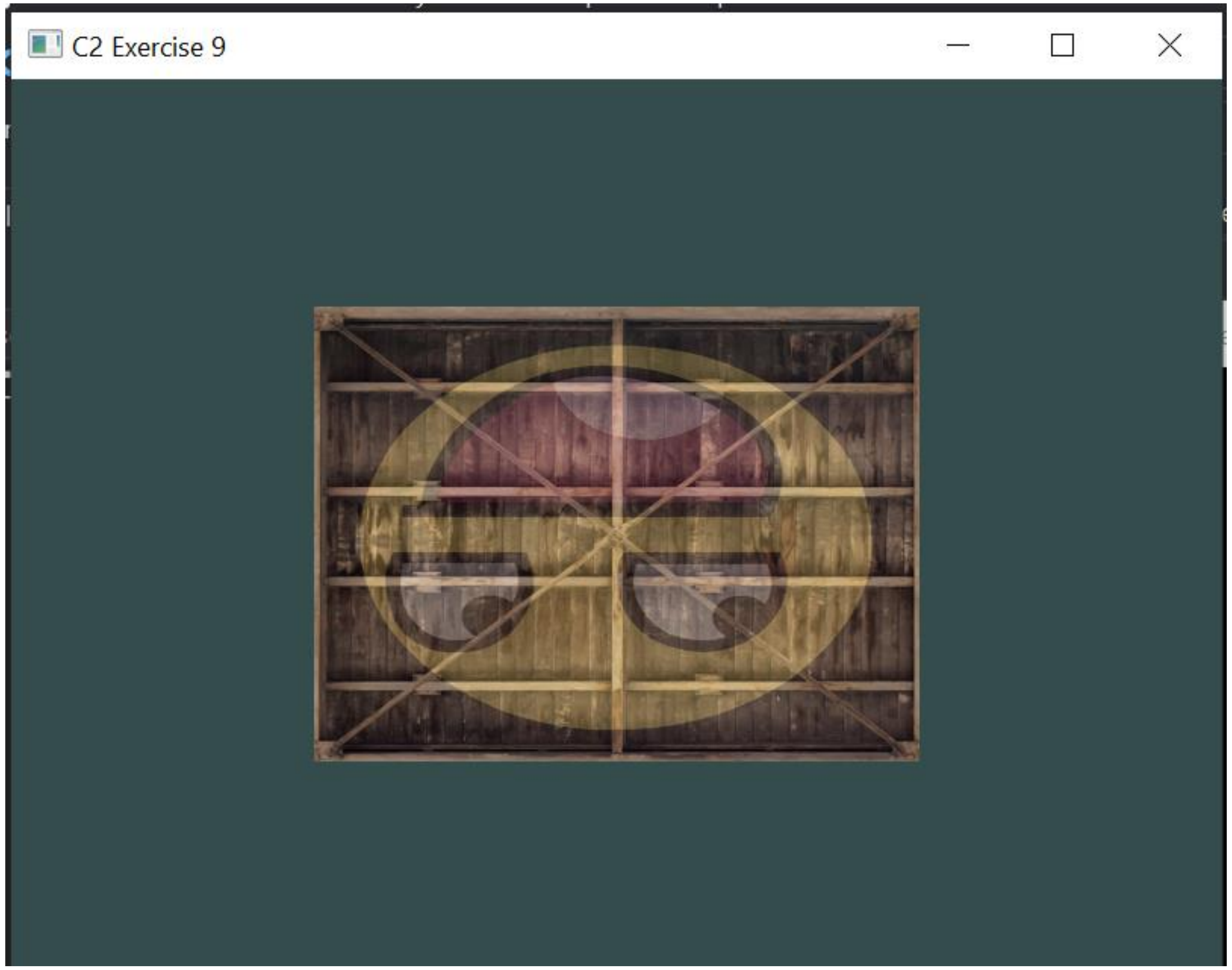
By setting the samplers via glUniform1i we make sure each uniform sampler corresponds to the proper texture unit.

Textures – Texture Units


Texture Objects
Texture Units
Sampler Variables in Fragment Shader
tex1
tex2
tex3
Texture Unit 0 (gl.TEXTURE0)
Texture Unit 1 (gl.TEXTURE1)
uniform sampler2D x;
uniform sampler2D y;
Association set up with:
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
Created with:
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . .);
Association set up with:
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);

Exercise 9:

Test the use of Texture Units in OpenGL using two texture figures.

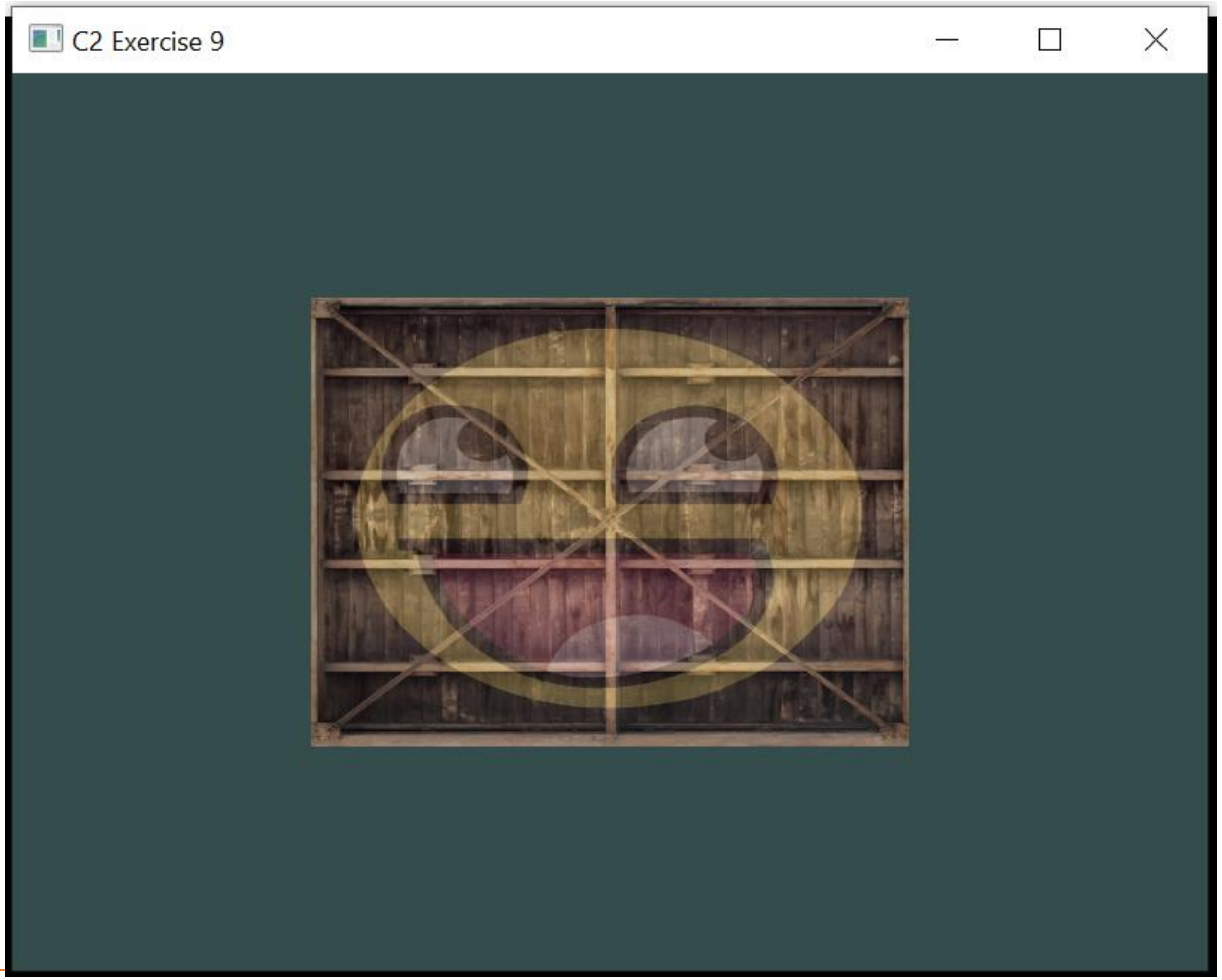


Textures – Texture Units

Exercise 9:
Test the use of Texture Units in OpenGL using two texture figures (C2_Exercise_9_TexureUnits.cpp).

- You probably noticed that the texture is flipped upside-down!
- This happens because OpenGL expects the 0.0 coordinate on the y-axis to be on the bottom side of the image, but images usually have 0.0 at the top of the y-axis.
- Luckily for us, stb_image.h can flip the y-axis during image loading by adding the following statment before loading any image:

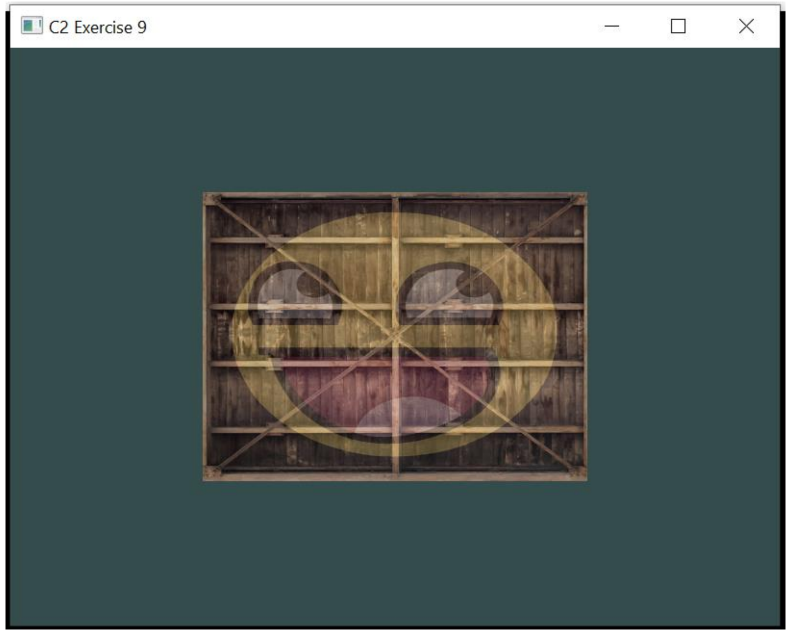
```
stbi_set_flip_vertically_on_load(true);
```

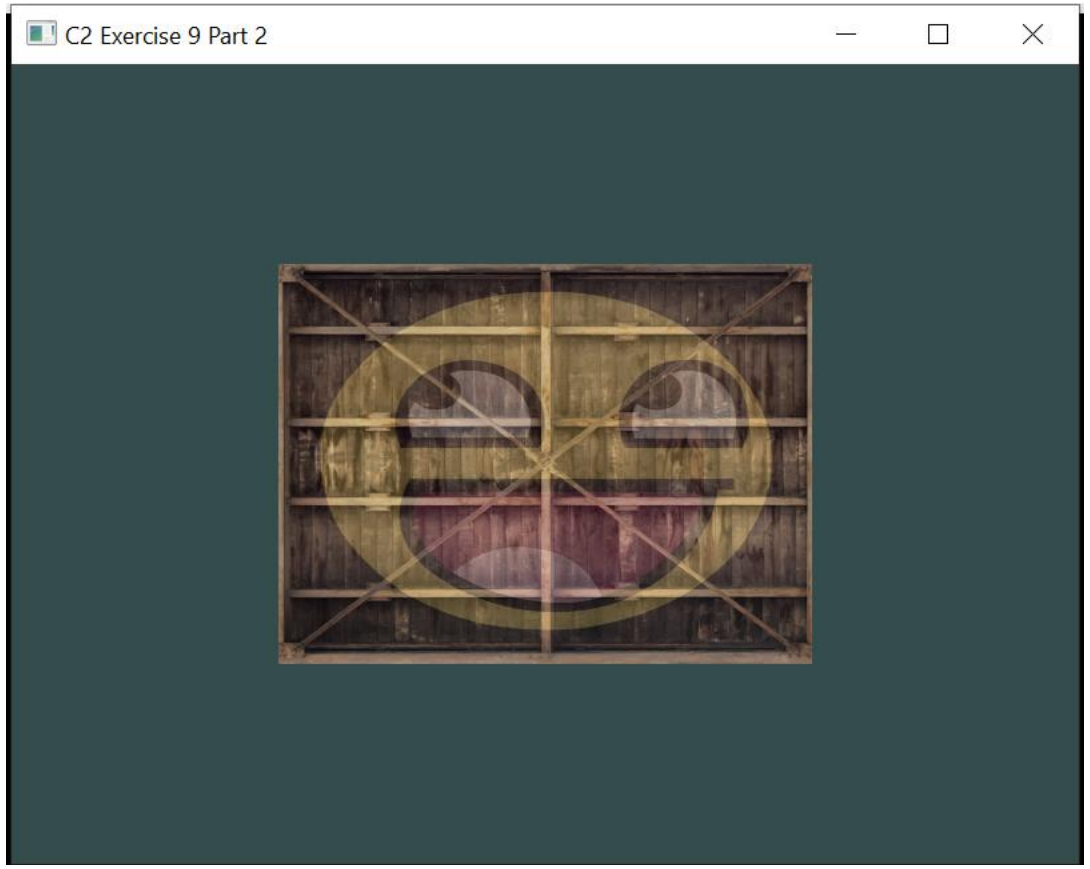


Textures – Texture Units

Exercise 9 Task 2:

Only the happy face looks in the other/reverse direction by changing the fragment shader.



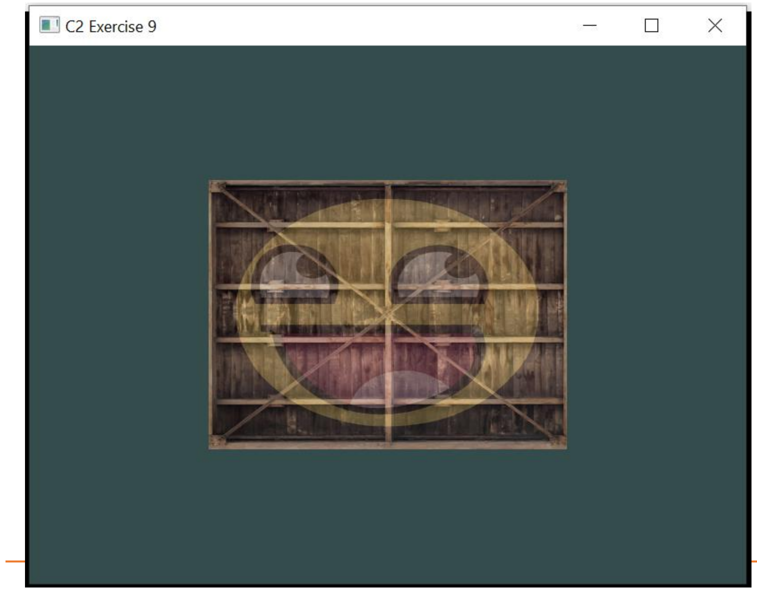


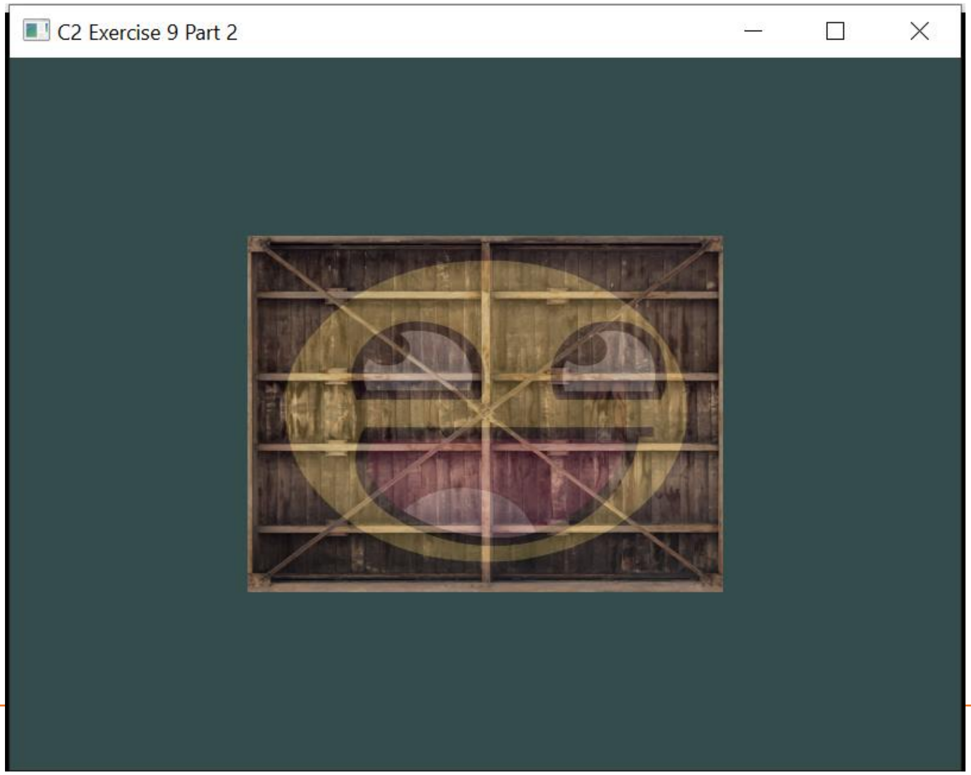
Textures – Texture Units

Exercise 9 Task 2:

Only the happy face looks in the other/reverse direction by changing the fragment shader.

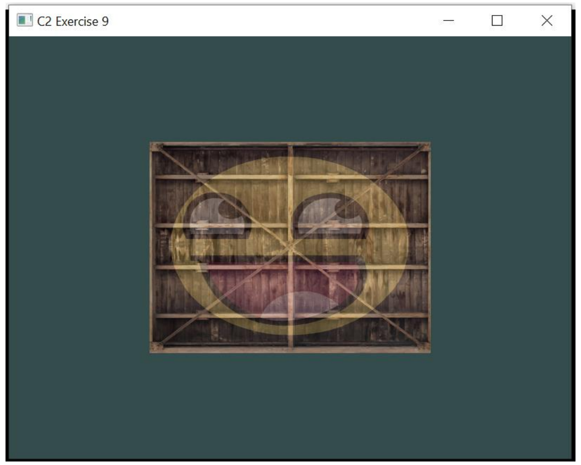
```
[…]
void main()
{
    FragColor = mix(texture(texture1, TexCoord), texture(texture2, vec2(1.0 - TexCoord.x, TexCoord.y)), 0.2);
}
```

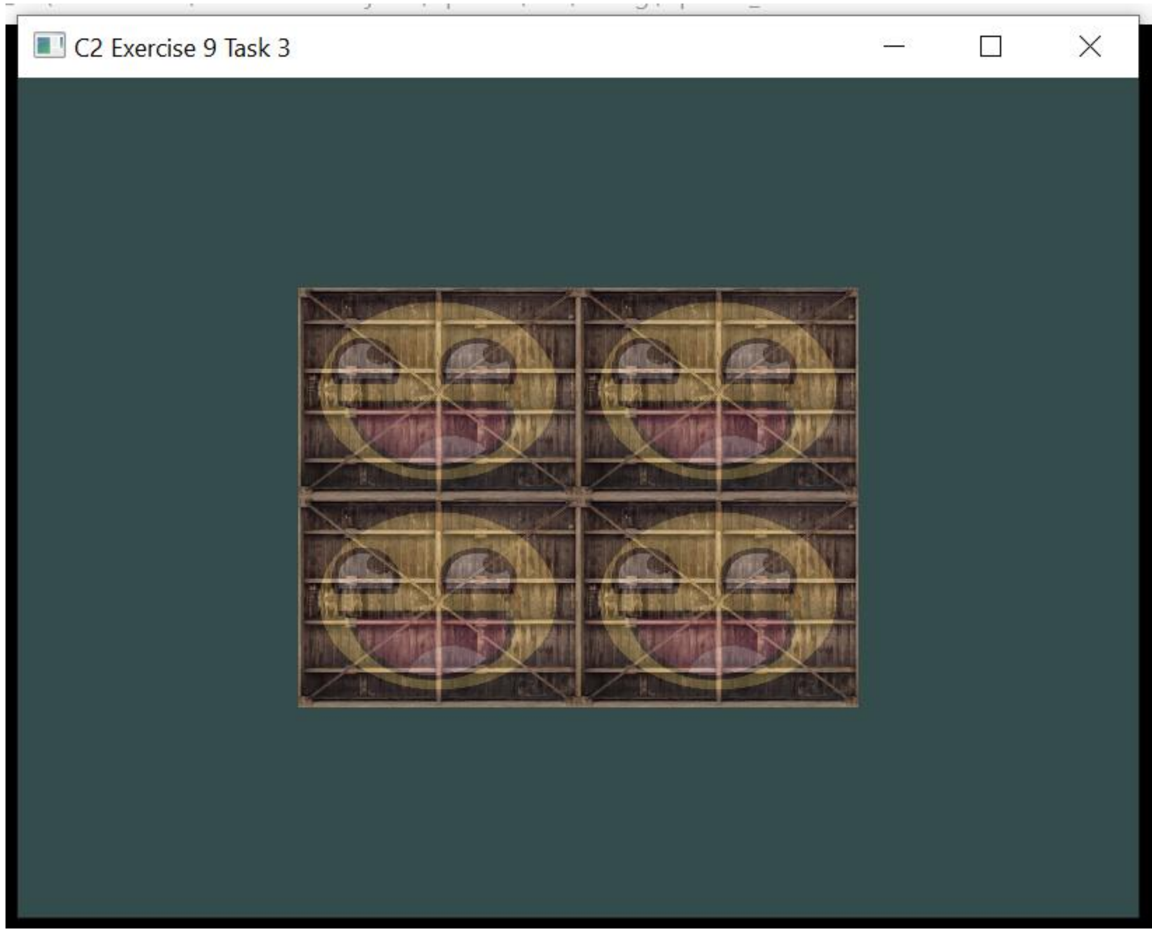




Textures – Texture Units

Exercise 9 Task 3:
Experiment with the different texture wrapping methods by specifying texture coordinates in the **range 0.0f to 2.0f** instead of 0.0f to 1.0f. See if you can display 4 smiley faces on a single container image clamped at its edge:.



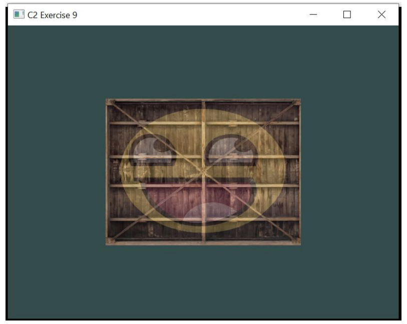


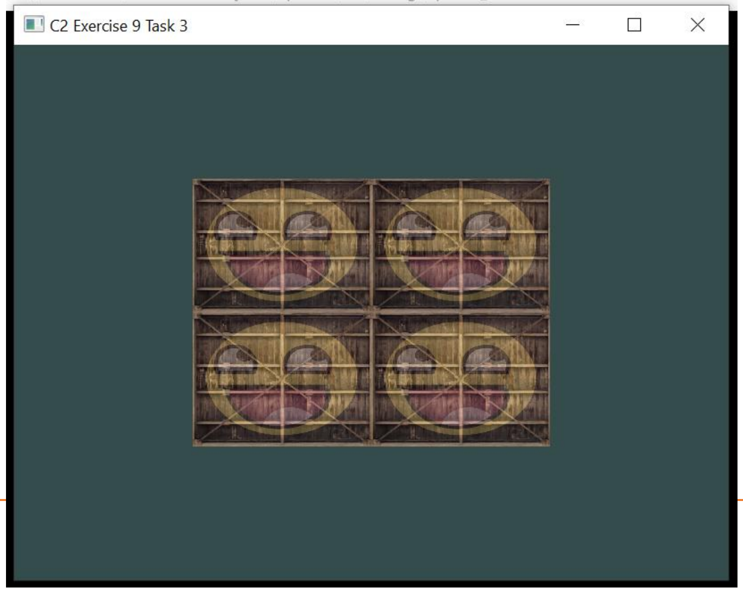
Textures – Texture Units

Exercise 9 Task 3:

Experiment with the different texture wrapping methods by specifying texture coordinates in the **range 0.0f to 2.0f** instead of 0.0f to 1.0f. See if you can display 4 smiley faces on a single container image clamped at its edge:.

```
float vertices[] = {
     // positions         // colors          // texture coords
      0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   2.0f, 2.0f, // top right
      0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   2.0f, 0.0f, // bottom right
     -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f, // bottom left
     -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 2.0f  // top left
  };
```

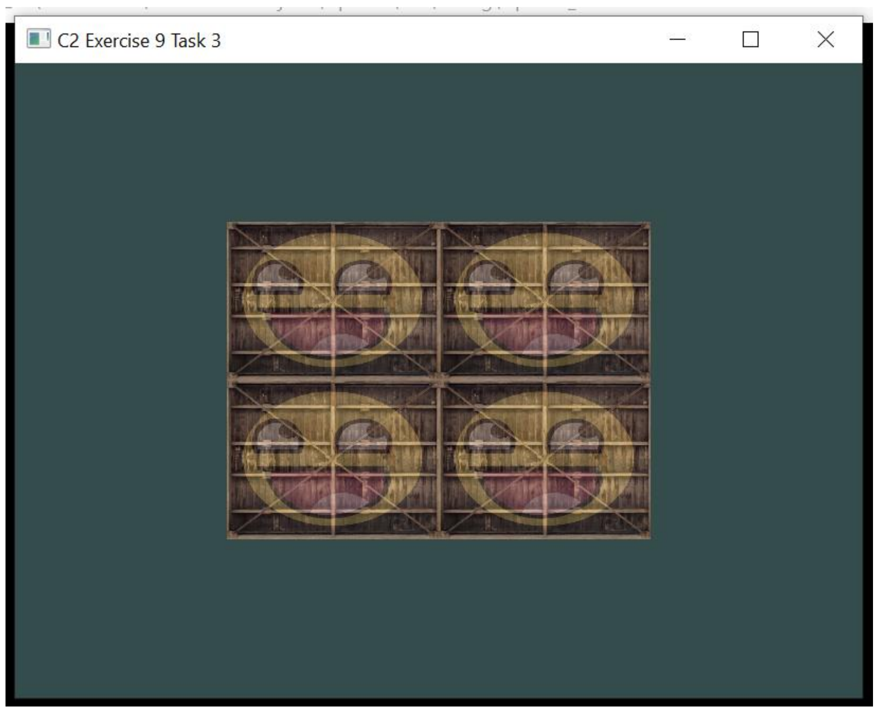


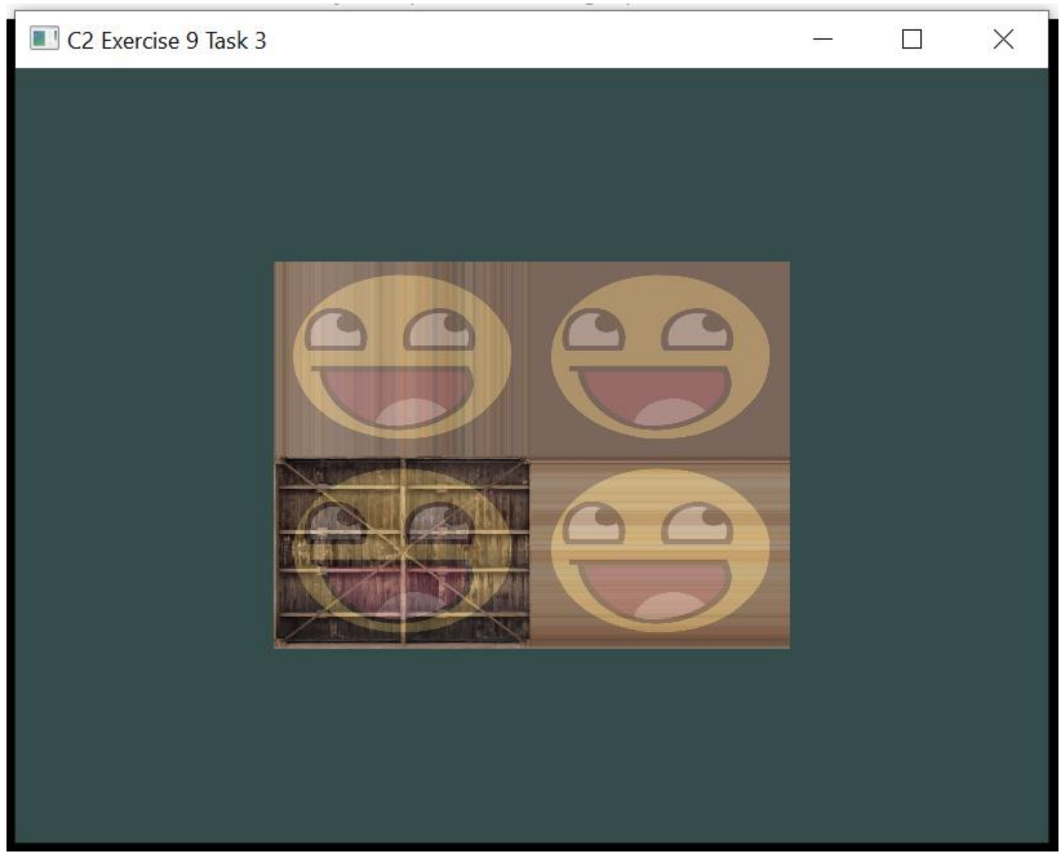


Textures – Texture Units

Exercise 9 Task 3:

(2) Experiment with the different texture wrapping methods by specifying texture coordinates in the **range 0.0f to 2.0f** instead of 0.0f to 1.0f. See if you can display 4 smiley faces on a single container image clamped at its edge:.



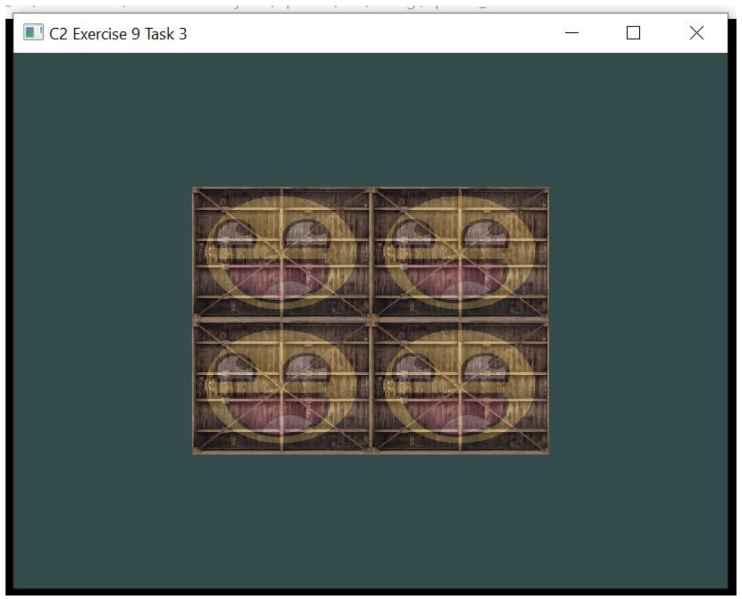


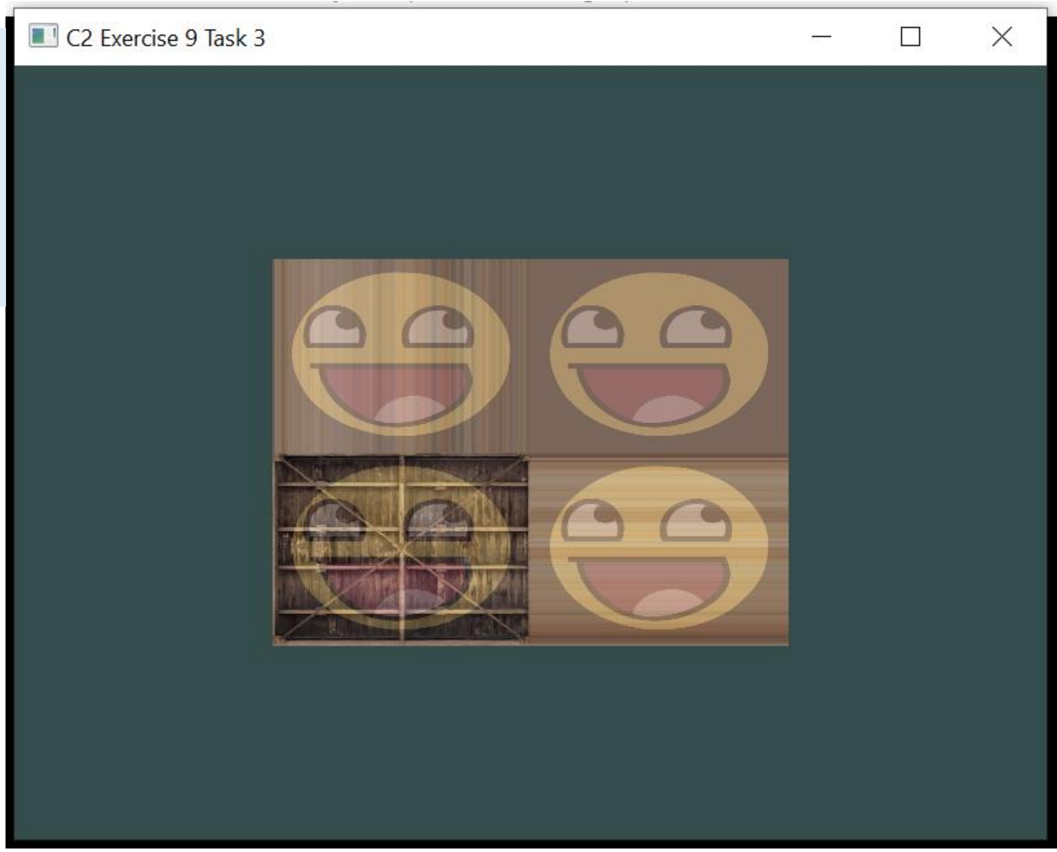
Textures – Texture Units

Exercise 9 Task 3:

(2) Experiment with the different texture wrapping methods by specifying texture coordinates in the **range 0.0f to 2.0f** instead of 0.0f to 1.0f. See if you can display 4 smiley faces on a single container image clamped at its edge:.

```
glGenTextures(1, &texture1);
  glBindTexture(GL_TEXTURE_2D, texture1);
  // set the texture wrapping parameters
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```



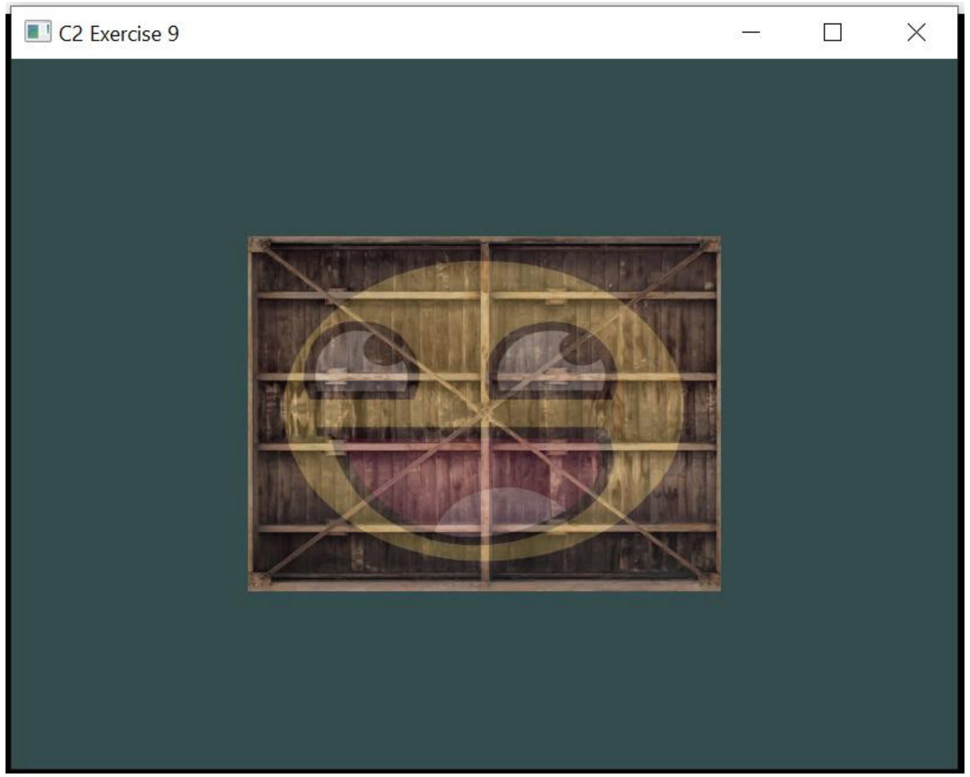


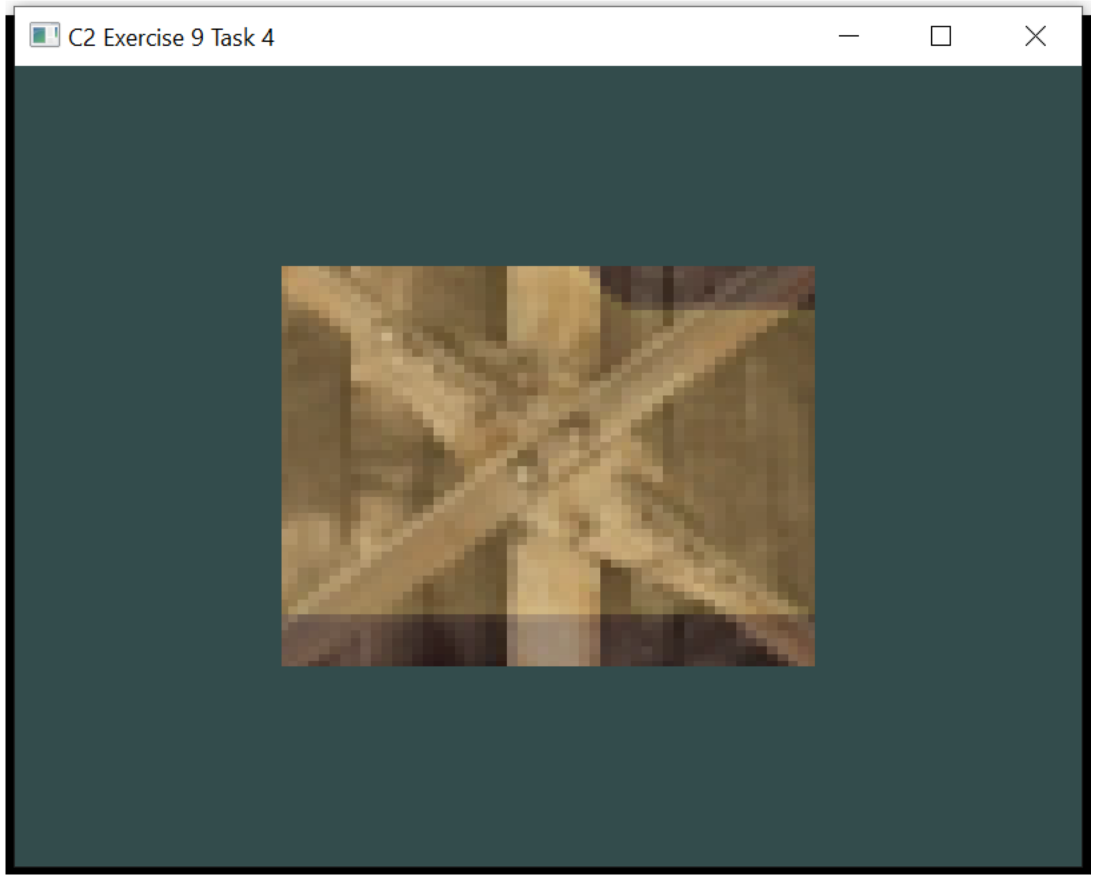
Please, experiment with other wrapping methods as well ☺

Textures – Texture Units

Exercise 9 Task 4:

Try to display only the center pixels of the texture image on the rectangle in such a way that the individual pixels are getting visible by changing the texture coordinates **(Zoom In Effect)**. Try to set the texture filtering method to **GL_NEAREST** to see the pixels more clearly



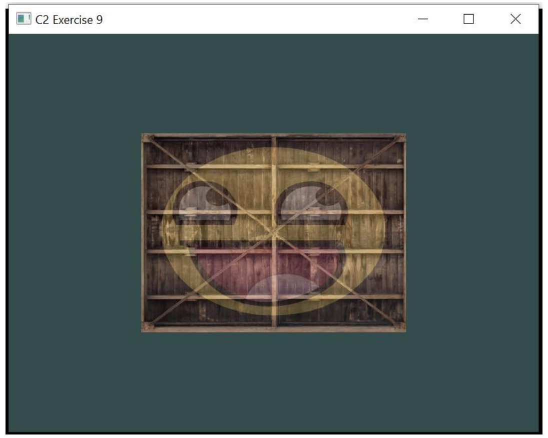


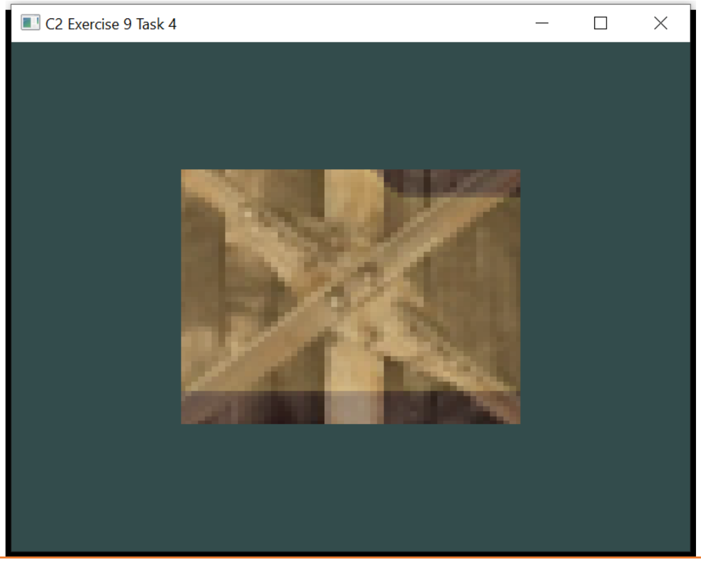
Textures – Texture Units

Exercise 9 Task 4:

Try to display only the center pixels of the texture image on the rectangle in such a way that the individual pixels are getting visible by changing the texture coordinates **(Zoom In Effect)**. Try to set the texture filtering method to **GL_NEAREST** to see the pixels more clearly

```
float vertices[] = {
     // positions          // colors           // texture coords (note that we changed them to 'zoom in' on our texture image)
      0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   0.55f, 0.55f, // top right
      0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   0.55f, 0.45f, // bottom right
     -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.45f, 0.45f, // bottom left
     -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.45f, 0.55f  // top left
  };
```

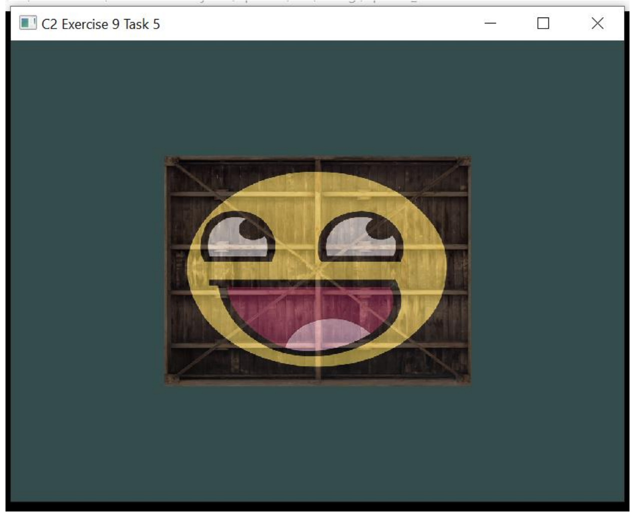


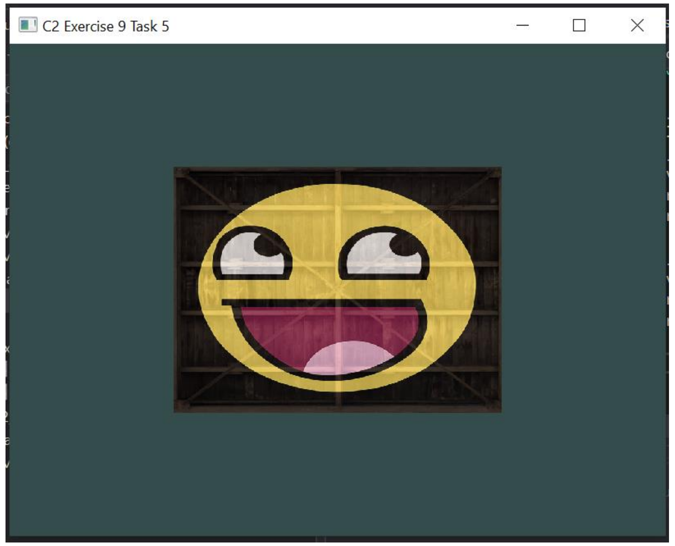


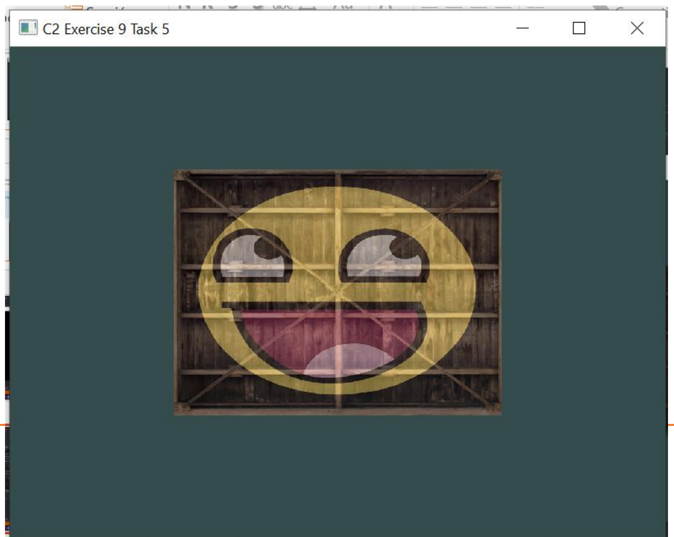
Textures – Texture Units

Exercise 9 Task 5:

Use a **uniform variable** as the mix function's **third parameter** to vary the amount the two textures are visible. Use the **up and down arrow keys** to change how much the container or the smiley face is visible:



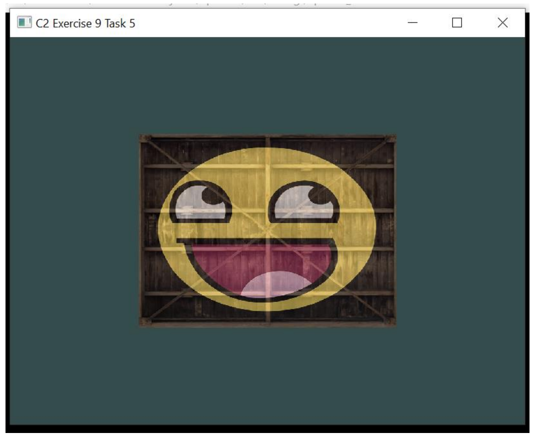


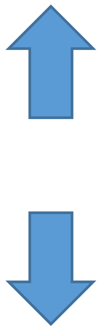


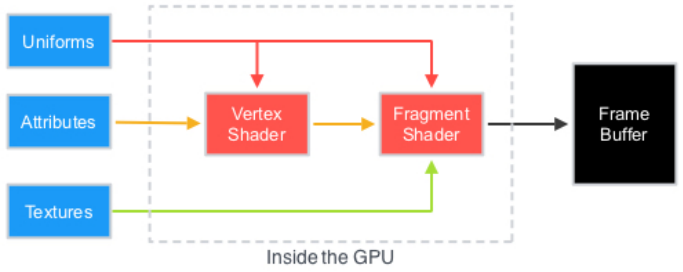
Textures – Texture Units

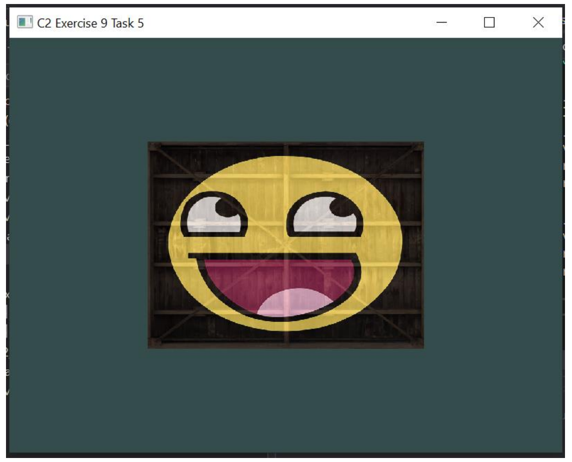
Exercise 9 Task 5:

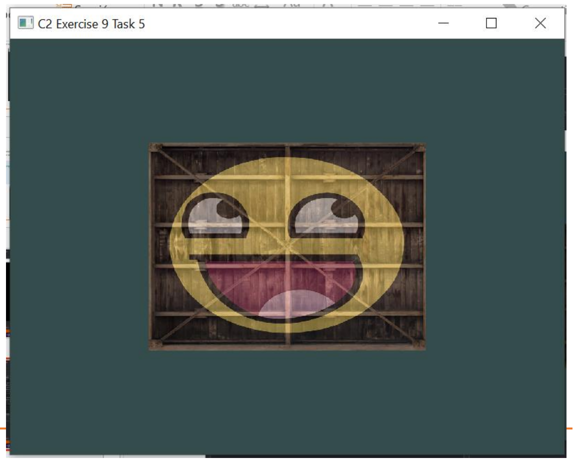
Use a **uniform variable** as the mix function's **third parameter** to vary the amount the two textures are visible. Use the **up and down arrow keys** to change how much the container or the smiley face is visible:











Textures – Texture Units

Exercise 9 Task 5:

Use a **uniform variable** as the mix function's **third parameter** to vary the amount the two textures are visible. Use the **up and down arrow keys** to change how much the container or the smiley face is visible:

Fragment Shader

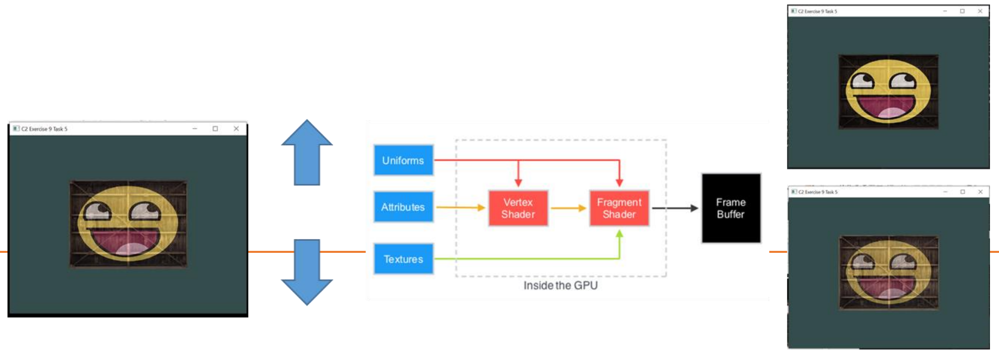
```
[…]
uniform float visible;
void main()
{
FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), visible);
}
```

Render Loop

```
float visibilyFactor = 0.5f;
[…]
while(){
   //bind textures
    […]
     glBindTexture(GL_TEXTURE_2D, texture2);
     ourShader.setFloat("visible", visibilyFactor);
     // render container
     ourShader.use();
     […]
}
```

Window Input

```
void processInput(GLFWwindow* window)
{
   if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
      glfwSetWindowShouldClose(window, true);
   if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS){
      visibilyFactor = visibilyFactor + 0.001f;
      if (visibilyFactor >= 1.0f)
         visibilyFactor = 1.0f; }
   if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS) {
      visibilyFactor = visibilyFactor - 0.001f;
      if (visibilyFactor <= 0.0f)
         visibilyFactor = 0.0f; }
}
```



Uniforms
Attributes
Textures
Vertex Shader
Fragment Shader
Frame Buffer
Inside the GPU