JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Computer Graphics

Lab 5: Texturing

Fork me on GitHub

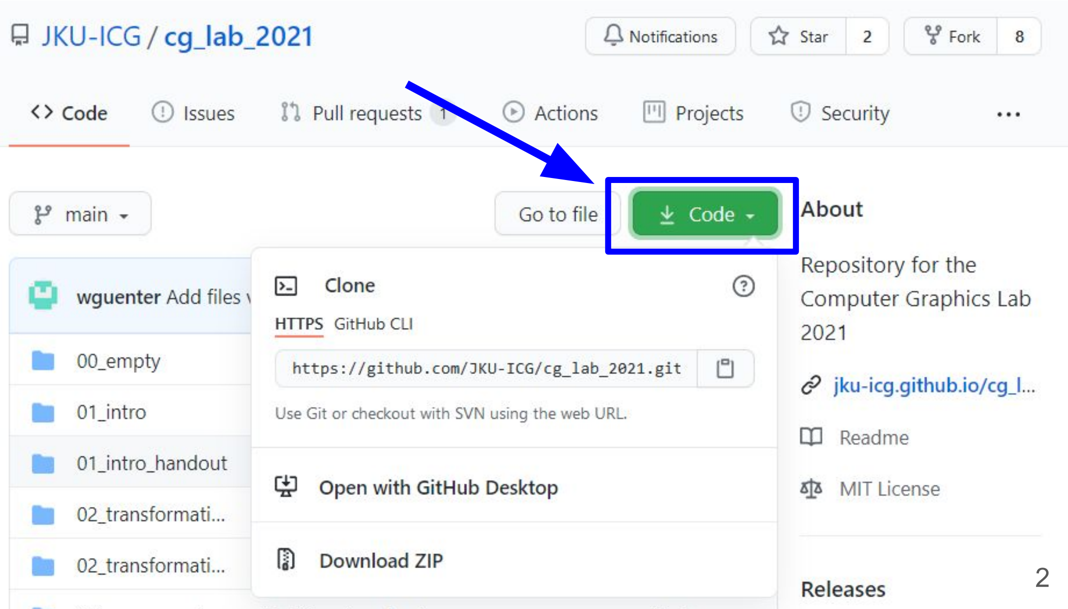
Dev Environment: Lab Package

Hosted on GitHub: https://github.com/jku-icg/cg_lab_2021

The repository will be updated during the lab with the new projects.

To get started (now):

1. Download the ZIP
2. Extract the folder
3. Open Visual Studio Code
4. Open `cg_lab_2021` folder (*File → Open*)
5. Click on **Go Live** button in lower right corner



```
https://github.com/JKU-ICG/cg_lab_2021.git
```

Use Git or checkout with SVN using the web URL.

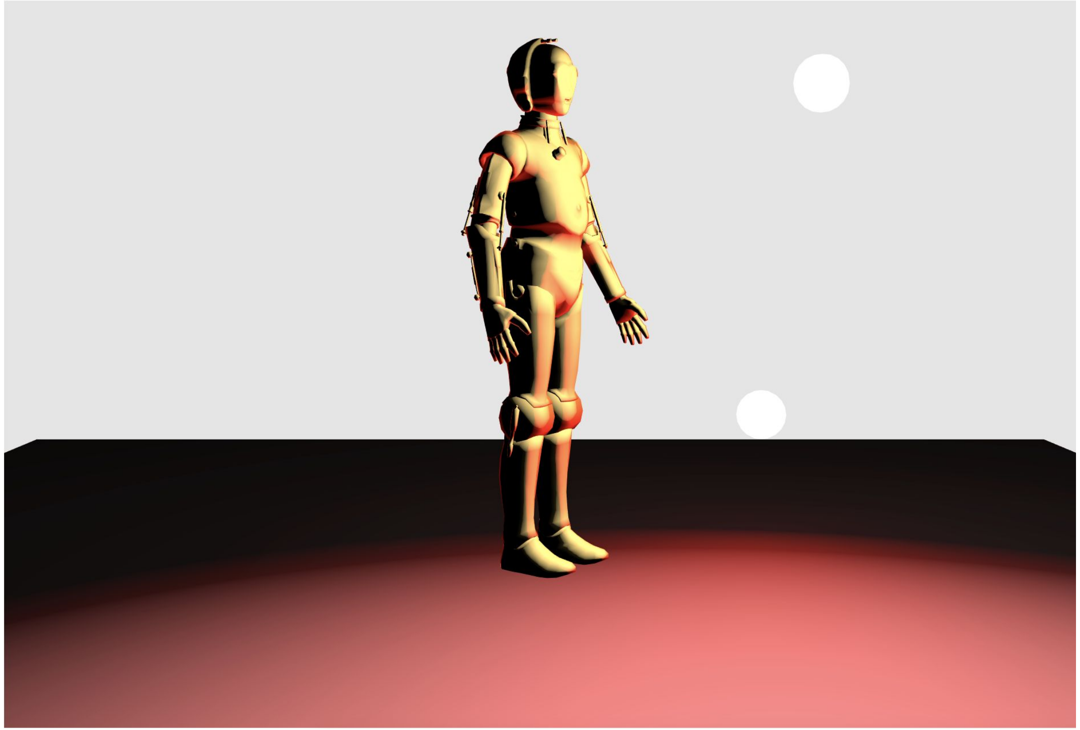
Repository for the Computer Graphics Lab 2021

Recap

Lab Project Specification

Illumination

0. Interaction
1. Static Phong Shader
2. New SG Node: Material
3. New SG Node: Light
4. Animated Light
5. Multiple Lights

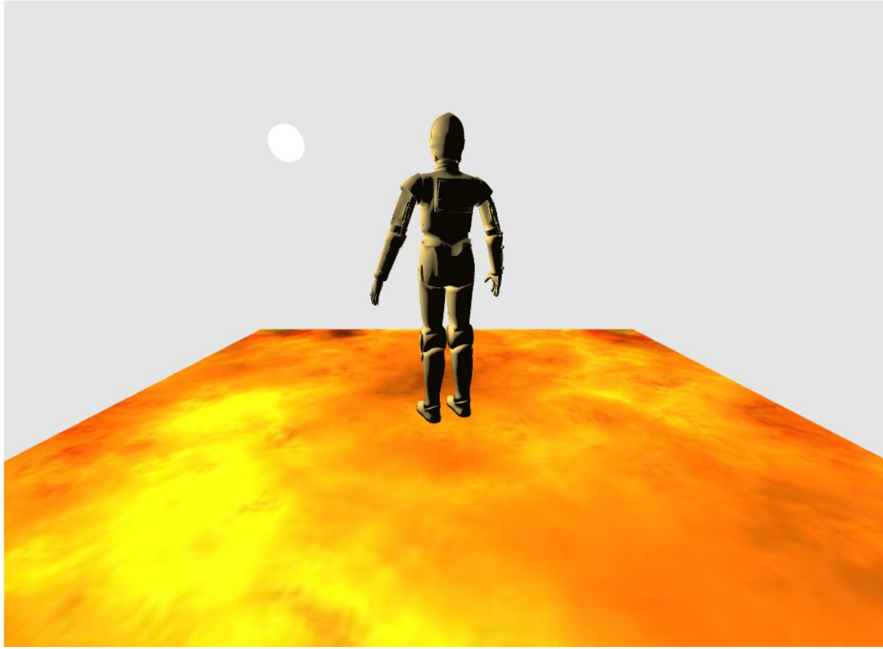


Solution is on GitHub.

Agenda for Today

Texting Basics

- Overview
- Setup Textures
- Texture Coordinates
- Textures in Shader
- Binding Textures
- Task: Simple Texturing
- Task: Integrate Texturing into Phong Shader
- Tasks: Texture Wrapping and Repeating

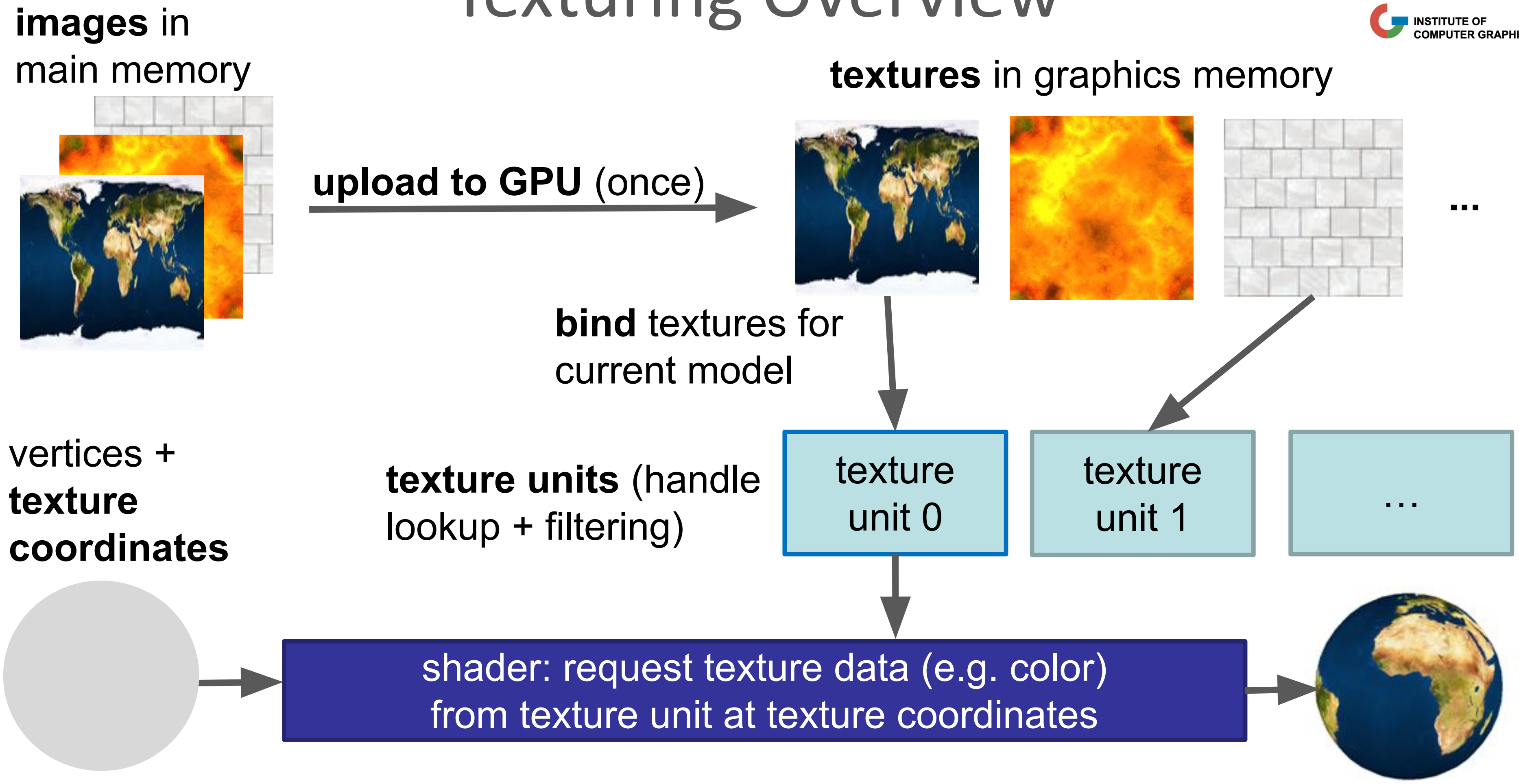


Rendering to Textures

- Multiple Render Passes
- Setup Framebuffer
- Task: Render to Texture

Extra Task: Animate Texture Coordinates

Texturing Overview



images in main memory

upload to GPU (once)

textures in graphics memory

...

bind textures for current model

vertices + **texture coordinates**

texture units (handle lookup + filtering)

texture unit 0

texture unit 1

...

shader: request texture data (e.g. color) from texture unit at texture coordinates

What Are Texture Units?

You can think of them as a piece of GPU hardware which performs fast image sampling. (not 100% correct!)

Main jobs

- Addressing, e.g. compute pixel index (132,12) from texture coordinate (0.342,0.012)
- Filtering, e.g. combine neighboring pixels if pixel index is not an integer

Limited number of texture units!

- Hardware and OpenGL version dependent
- Per shader stage, e.g. 4
- Total, e.g. 4*2=8 (vertex + fragment shader stage)
- Limits number of textures which can be used simultaneously in shader

BUT (almost) unlimited number of textures

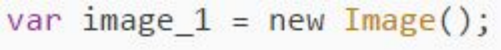
- Limited only by graphics memory size

Loading Images

Done in framework during resource loading!

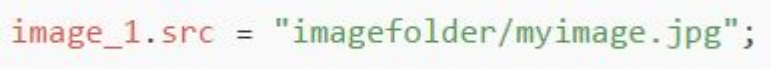
Basic steps:

Allocate JavaScript “Image” object:



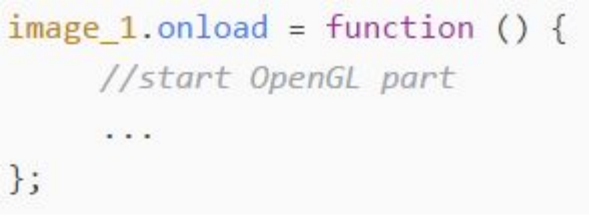
```
var image_1 = new Image();
```

Set image URL:



```
image_1.src = "imagefolder/myimage.jpg";
```

Wait until image is loaded:

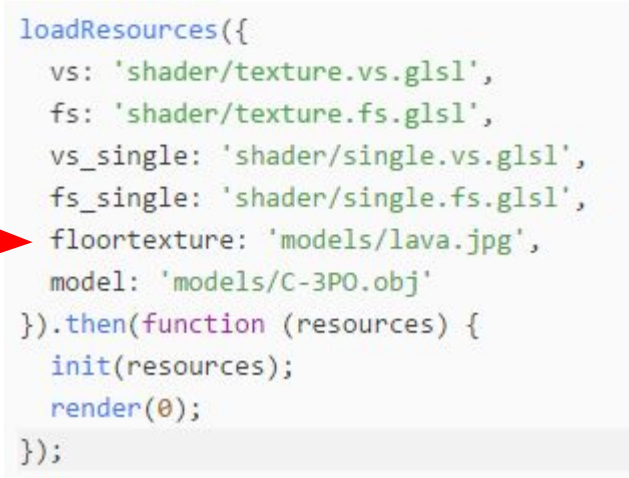


```
image_1.onload = function () {
    //start OpenGL part
    ...
};
```

See framework for more details.

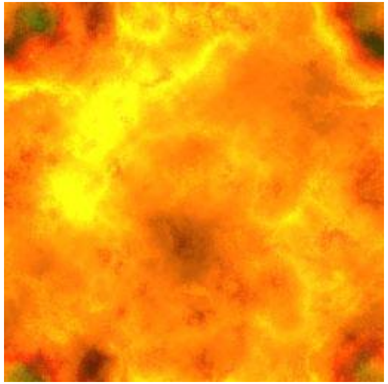
Recap: Loading Resources

Loading images with our framework is simple:



```
loadResources({
  vs: 'shader/texture.vs.glsl',
  fs: 'shader/texture.fs.glsl',
  vs_single: 'shader/single.vs.glsl',
  fs_single: 'shader/single.fs.glsl',
  floortexture: 'models/lava.jpg',
  model: 'models/C-3PO.obj'
}).then(function (resources) {
  init(resources);
  render(0);
});
```

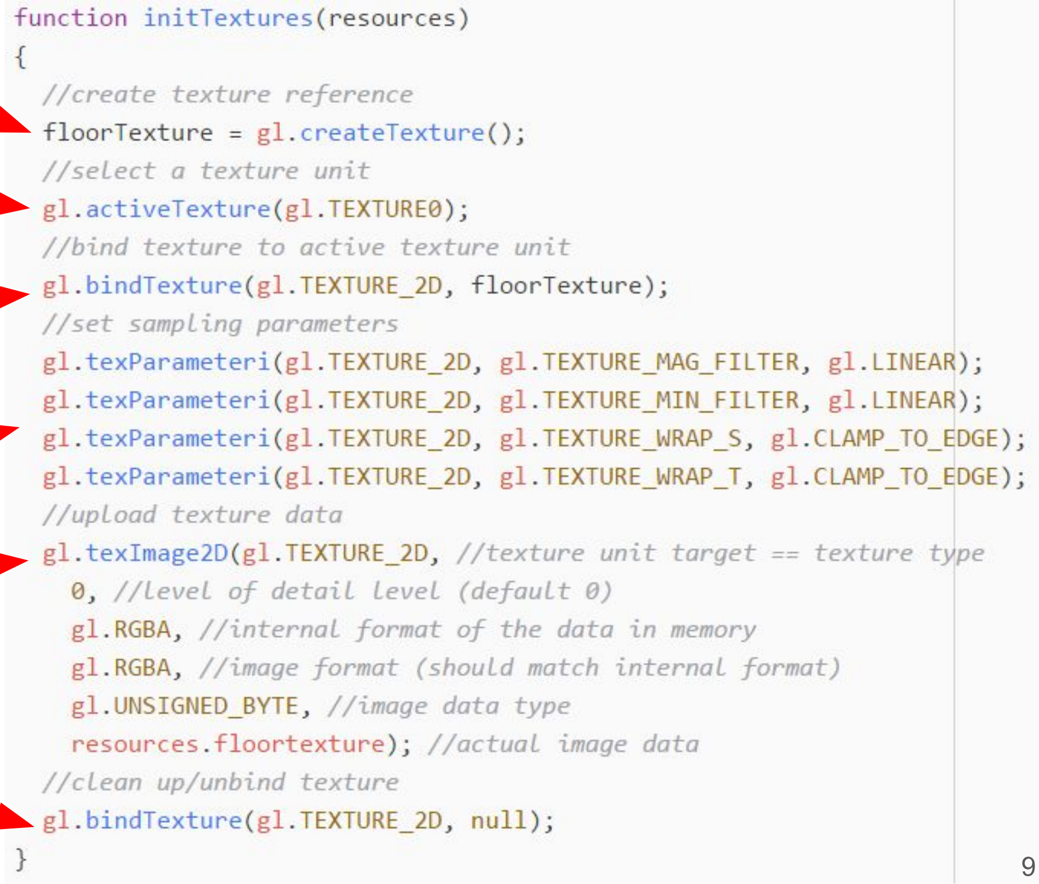
Access: `resources.floortexture`



`lava.jpg`

Code: Initialize Textures

1. create texture object
2. choose any texture unit
3. bind to texture unit
4. set sampling parameters
5. upload data to GPU
6. unbind texture

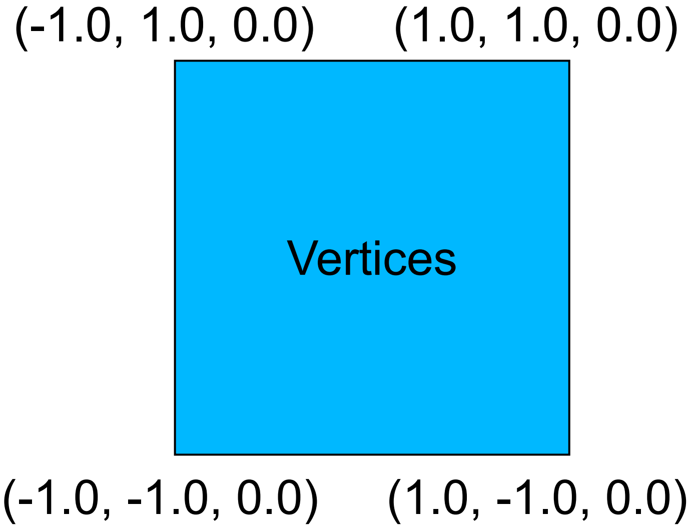


```
function initTextures(resources)
{
  //create texture reference
  floorTexture = gl.createTexture();
  //select a texture unit
  gl.activeTexture(gl.TEXTURE0);
  //bind texture to active texture unit
  gl.bindTexture(gl.TEXTURE_2D, floorTexture);
  //set sampling parameters
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
  //upload texture data
  gl.texImage2D(gl.TEXTURE_2D, //texture unit target == texture type
    0, //level of detail level (default 0)
    gl.RGBA, //internal format of the data in memory
    gl.RGBA, //image format (should match internal format)
    gl.UNSIGNED_BYTE, //image data type
    resources.floortexture); //actual image data
  //clean up/unbind texture
  gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Texture Coordinates

Mapping a texture onto a shape will be done by providing texture coordinates for every vertex.

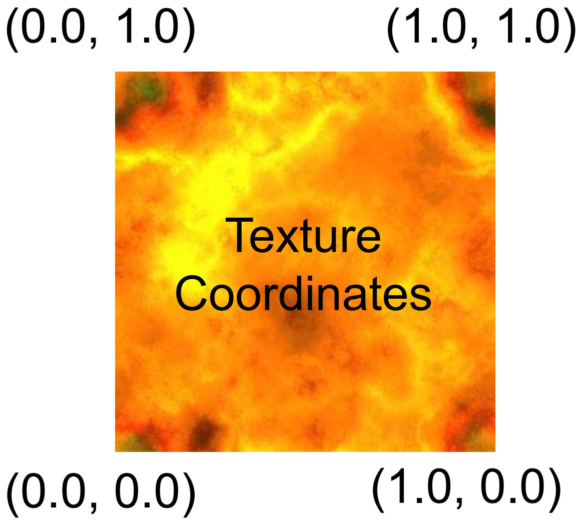
Texture coordinates for a simple quad:



(-1.0, 1.0, 0.0) (1.0, 1.0, 0.0)

Vertices

(-1.0, -1.0, 0.0) (1.0, -1.0, 0.0)

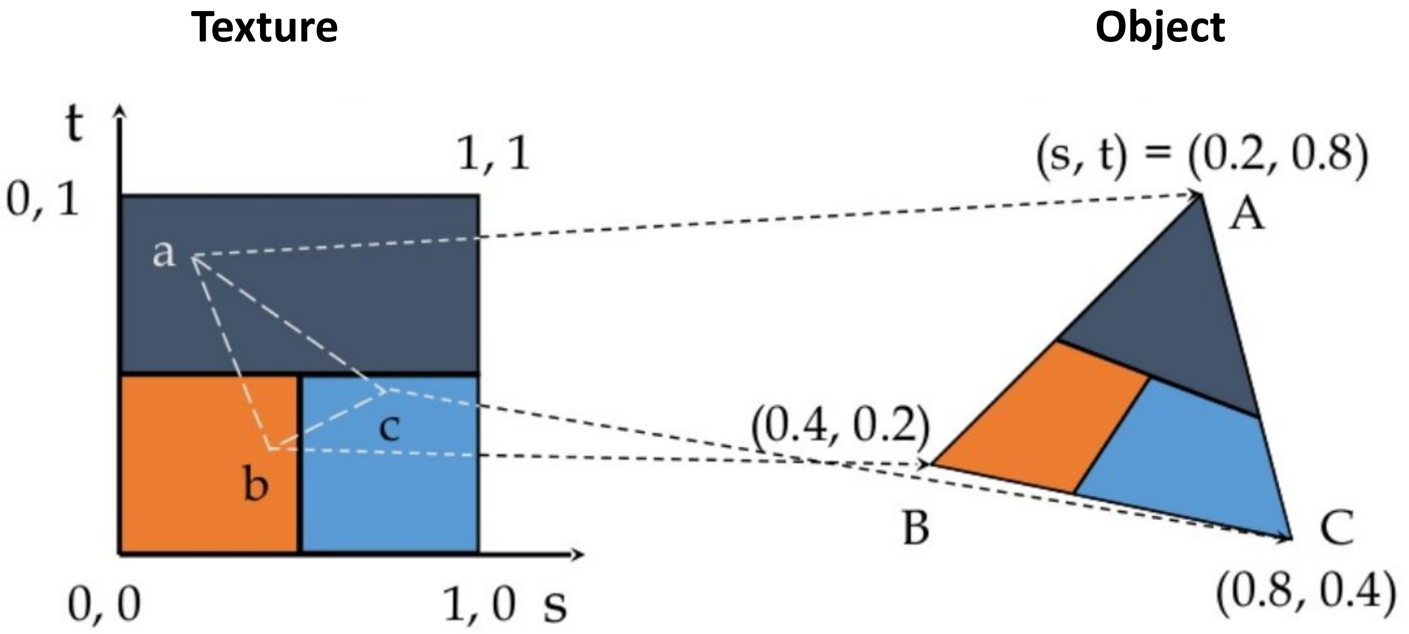


(0.0, 1.0) (1.0, 1.0)

Texture Coordinates

(0.0, 0.0) (1.0, 0.0)

Mapping a Texture



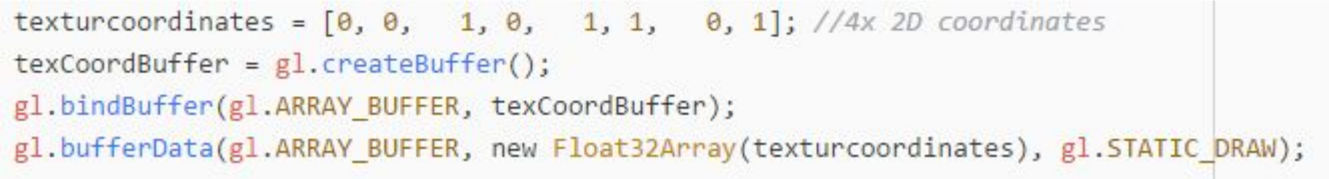
https://www.slideshare.net/SyedZaidIrshad/opengl-texture-mapping

Recap: Buffers and Attributes

Texture coordinates are defined per vertex via a buffer

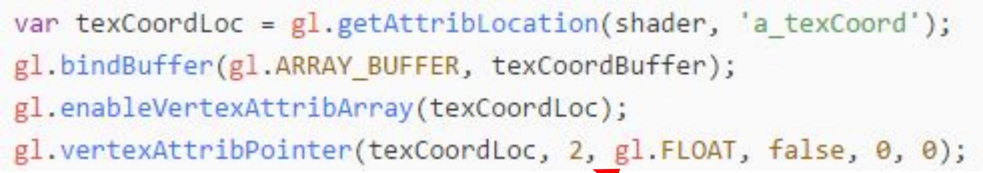
Same as vertex position, color, normal, …

Define coordinates + buffer:



```
texturcoordinates = [0, 0,   1, 0,   1, 1,   0, 1]; //4x 2D coordinates
texCoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, texCoordBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(texturcoordinates), gl.STATIC_DRAW);
```

Pass buffer to shader attribute (`a_texCoord`):



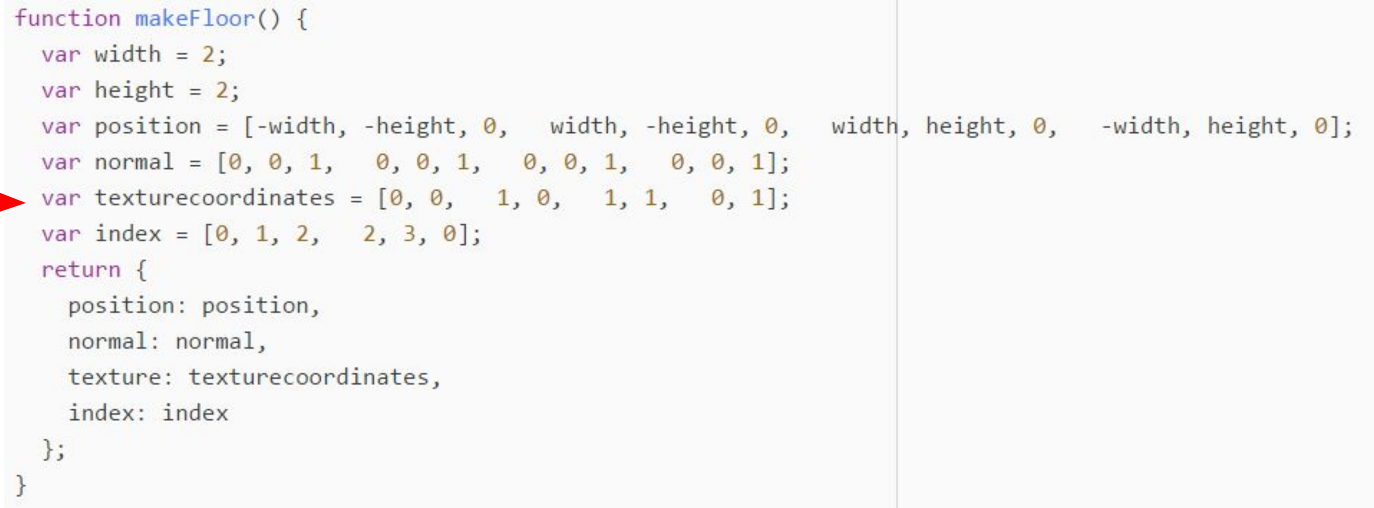
```
var texCoordLoc = gl.getAttribLocation(shader, 'a_texCoord');
gl.bindBuffer(gl.ARRAY_BUFFER, texCoordBuffer);
gl.enableVertexAttribArray(texCoordLoc);
gl.vertexAttribPointer(texCoordLoc, 2, gl.FLOAT, false, 0, 0);
```

remember? 2 == 2D vectors

See framework for more details

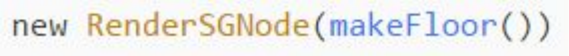
Code: Texture Coordinates for Floor

Define texture coordinates using our framework:



```
function makeFloor() {
  var width = 2;
  var height = 2;
  var position = [-width, -height, 0,   width, -height, 0,   width, height, 0,   -width, height, 0];
  var normal = [0, 0, 1,   0, 0, 1,   0, 0, 1,   0, 0, 1];
  var texturecoordinates = [0, 0,   1, 0,   1, 1,   0, 1];
  var index = [0, 1, 2,   2, 3, 0];
  return {
    position: position,
    normal: normal,
    texture: texturecoordinates,
    index: index
  };
}
```

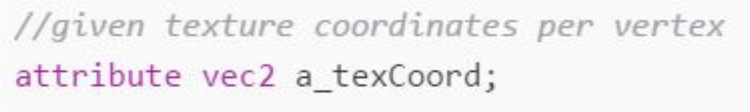
Assign to render node:



```
new RenderSGNode(makeFloor())
```

Texture Coordinates in Shader

Texture coordinates are bound to shader attributes:



```
//given texture coordinates per vertex
attribute vec2 a_texCoord;
```

Note: Our framework assigns texture coordinates to the attribute named `a_texCoord`.

Remember: Attributes are only available in vertex shader.
Have to be passed on via “varying” variables to fragment shader if required!

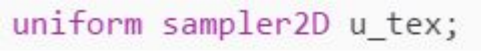
Texture Unit in Shader

Access to texture units via sampler variables

They don’t change per vertex ⟶ `uniform`

Different sampler type for each texture type: 2D ⟶ `sampler2D`

Define sampler for 2D textures:



```
uniform sampler2D u_tex;
```

Texture Lookup in Shader

Texture Lookup:

Get color of texture bound to sampler at defined coordinates.

Function:

```
texture2D(sampler,texture coordinates)
```

Input:

sampler (type: sampler2D)

texture coordinates (type: vec2)

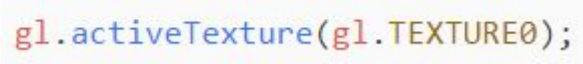
Returns:

color (type: vec4)

Bind Texture to Texture Unit

Before rendering a textured object we have to:

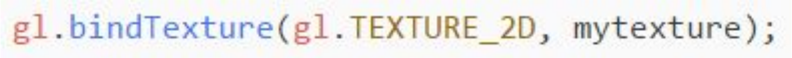
1. Select/activate a texture unit:



```
gl.activeTexture(gl.TEXTURE0);
```

Hint: Add texture unit number to `gl.TEXTURE0` to select a different unit!

2. Bind desired texture to selected texture unit:



```
gl.bindTexture(gl.TEXTURE_2D, mytexture);
```

Note: `gl.TEXTURE_2D` specifies the texture target = texture type

3. Assign texture unit number to sampler variable in shader.

Use `gl.uniform1i`(...) to set an integer to a uniform shader variable.

Hint: Do NOT use the texture unit constant (e.g. `gl.TEXTURE0`) in this case.

These are the duties of our `TextureSGNode` scene graph node!

Task 1: Simple Texturing

Goal:

Put texture on floor

Given:

Floor scenegraph node (`floor`)

Partially implemented texture node (`TextureSGNode`)

Partially implemented shaders (`texture.vs.glsl, texture.fs.glsl`)

Texture coordinates in vertex shader (`a_texCoord`)

Loaded and initialized texture (`floorTexture`)

Have a look at those code parts! Ask if you don't understand them.

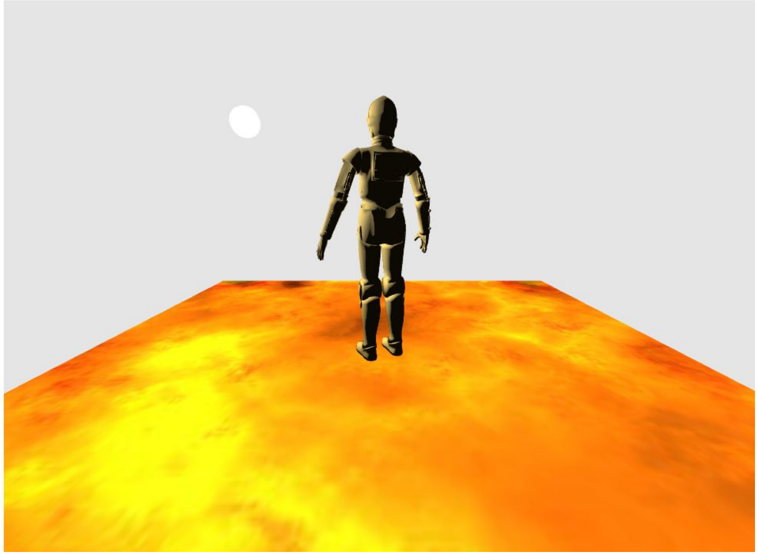
Task 1: Simple Texturing

Tasks in shaders:

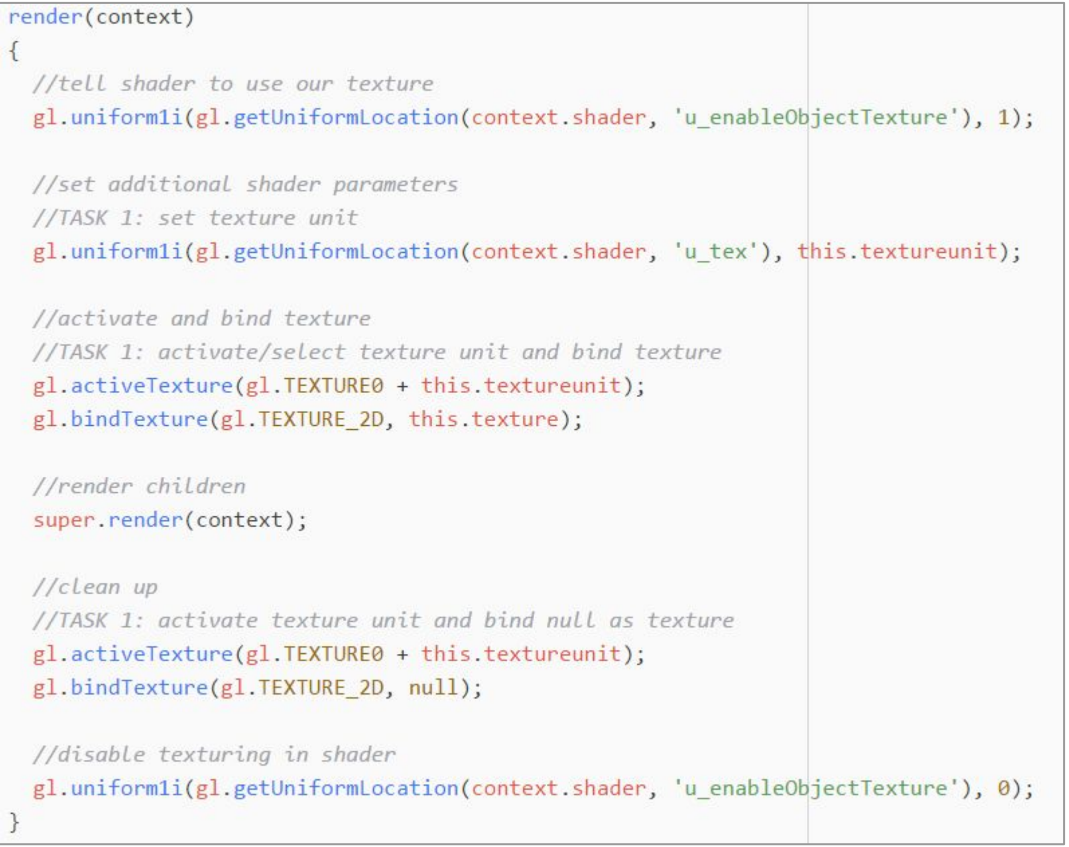
- Pass texture coordinates as varying to fragment shader
- Define sampler variable
- Do texture lookup in fragment shader (main)
  - Assign result to `gl_FragColor`

Tasks in main.js:

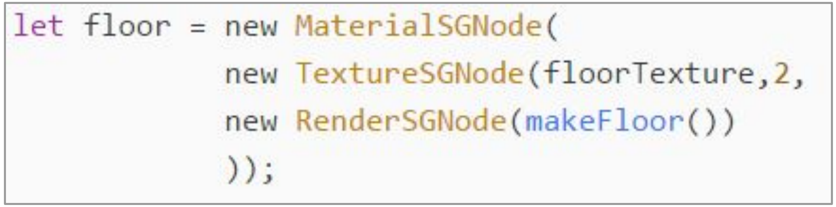
- Finish `TextureSGNode` implementation
  - Pass texture unit number to shader
  - Bind texture to texture unit
  - Clean up texture unit assignment
- Apply `TextureSGNode` to floor in scene graph
  - Use texture `floorTexture`
  - Use texture unit number 2



Task 1: main.js Solution



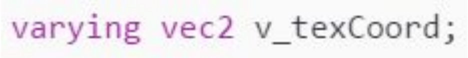
```
render(context)
{
  //tell shader to use our texture
  gl.uniform1i(gl.getUniformLocation(context.shader, 'u_enableObjectTexture'), 1);

  //set additional shader parameters
  //TASK 1: set texture unit
  gl.uniform1i(gl.getUniformLocation(context.shader, 'u_tex'), this.textureunit);

  //activate and bind texture
  //TASK 1: activate/select texture unit and bind texture
  gl.activeTexture(gl.TEXTURE0 + this.textureunit);
  gl.bindTexture(gl.TEXTURE_2D, this.texture);

  //render children
  super.render(context);

  //clean up
  //TASK 1: activate texture unit and bind null as texture
  gl.activeTexture(gl.TEXTURE0 + this.textureunit);
  gl.bindTexture(gl.TEXTURE_2D, null);

  //disable texturing in shader
  gl.uniform1i(gl.getUniformLocation(context.shader, 'u_enableObjectTexture'), 0);
}
```



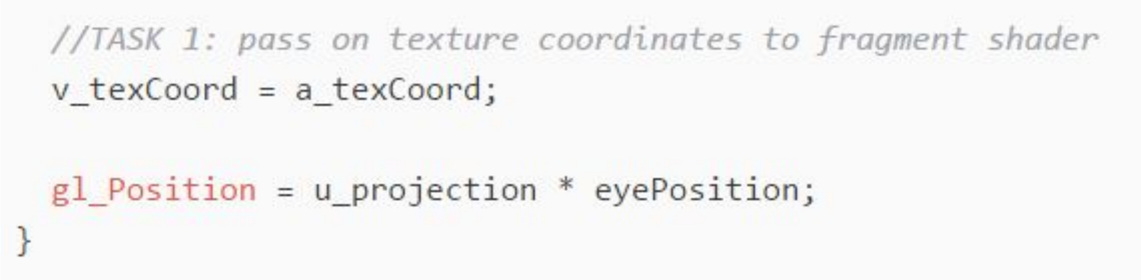
```
let floor = new MaterialSGNode(
            new TextureSGNode(floorTexture,2,
            new RenderSGNode(makeFloor())
            ));
```

Task 1: Shader Solution

vertex shader:

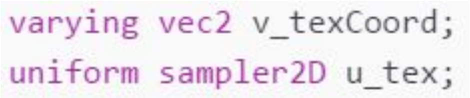


```
varying vec2 v_texCoord;
```

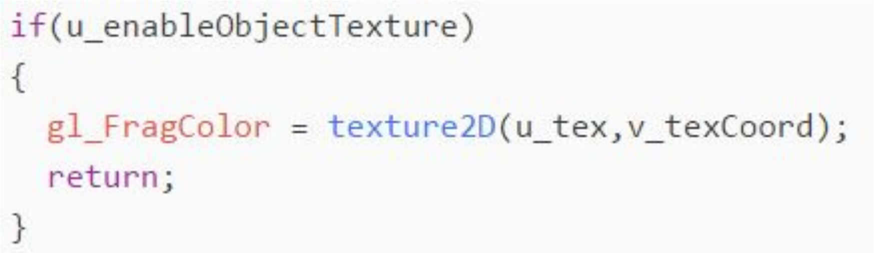


```
  //TASK 1: pass on texture coordinates to fragment shader
  v_texCoord = a_texCoord;

  gl_Position = u_projection * eyePosition;
}
```

fragment shader:



```
varying vec2 v_texCoord;
uniform sampler2D u_tex;
```



```
if(u_enableObjectTexture)
{
  gl_FragColor = texture2D(u_tex,v_texCoord);
  return;
}
```

Task 2: Phong Shader Integration

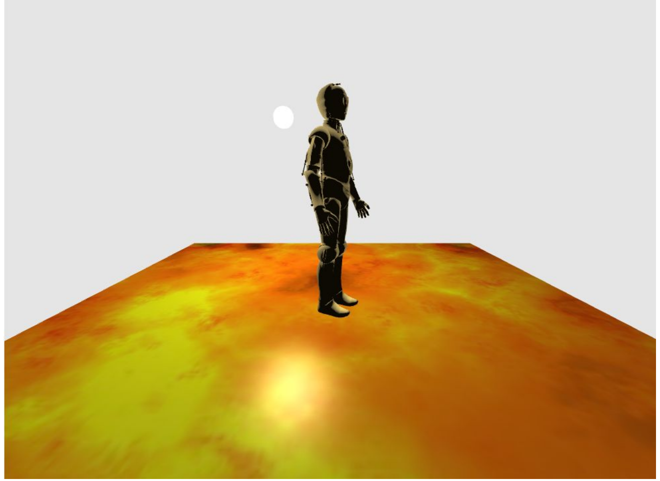
Goal:

The light should influence our textured object.

Tasks:

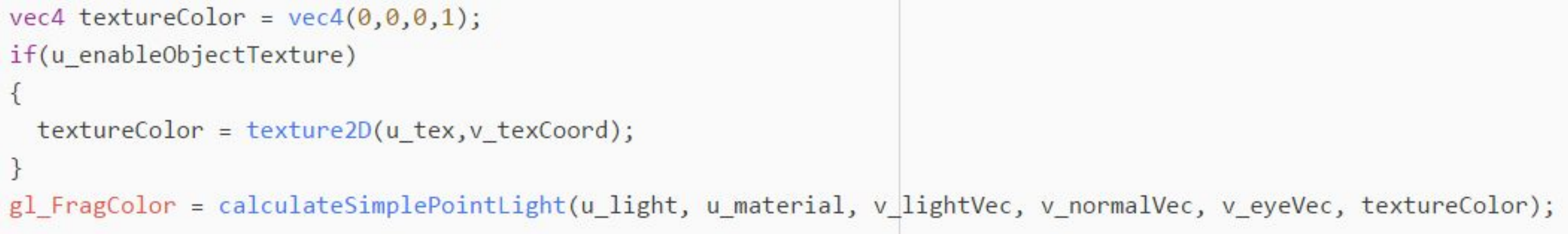
Pass texture color to `calculateSimplePointLight` function (instead of setting `gl_FragColor`)
Hint: use `textureColor` variable

Replace diffuse and ambient material color with texture color



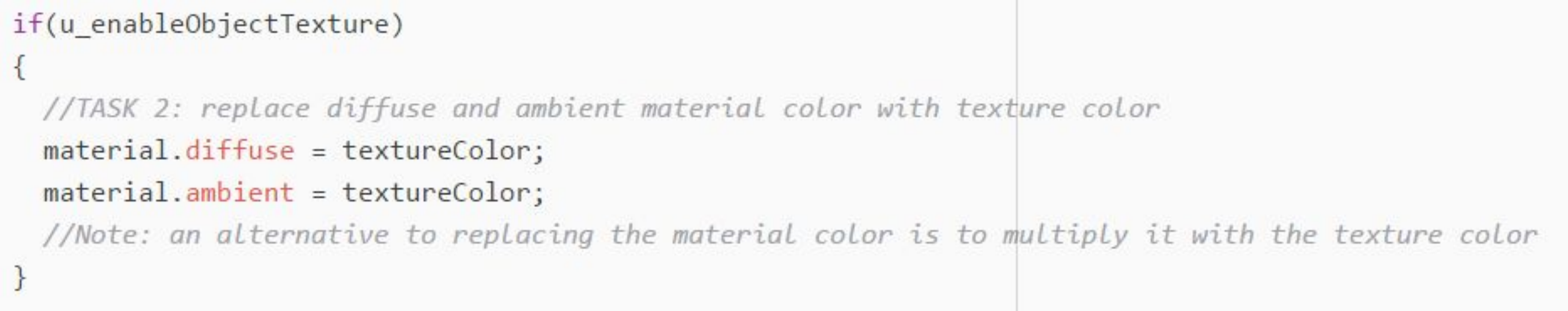
Task 2: Phong Shader Solution

`main` shader function:



```
vec4 textureColor = vec4(0,0,0,1);
if(u_enableObjectTexture)
{
  textureColor = texture2D(u_tex,v_texCoord);
}
gl_FragColor = calculateSimplePointLight(u_light, u_material, v_lightVec, v_normalVec, v_eyeVec, textureColor);
```

`calculateSimplePointLight` function:



```
if(u_enableObjectTexture)
{
  //TASK 2: replace diffuse and ambient material color with texture color
  material.diffuse = textureColor;
  material.ambient = textureColor;
  //Note: an alternative to replacing the material color is to multiply it with the texture color
}
```

Task 3: Modify Texture Coordinates

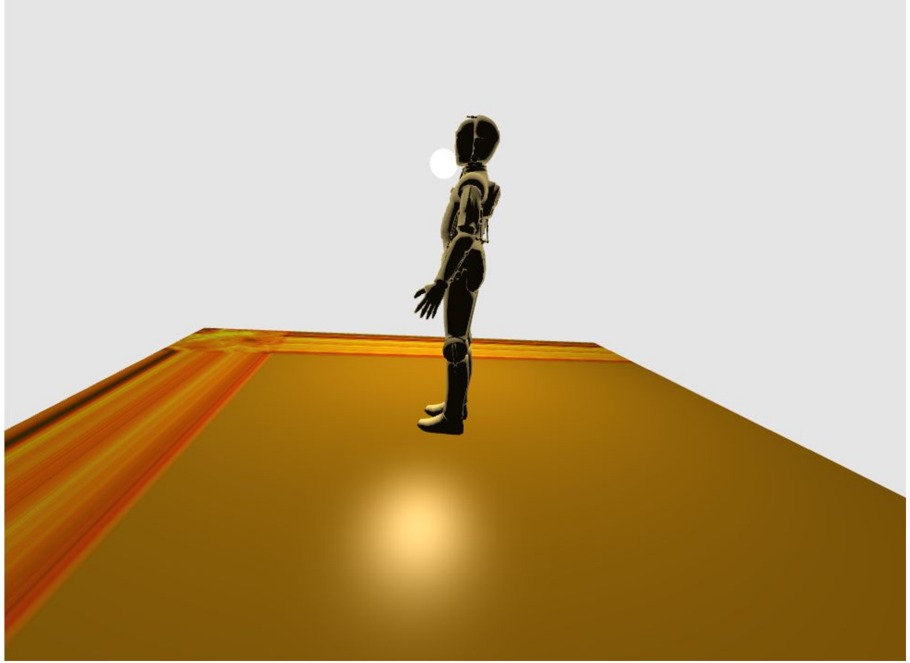
Goal:

The texture should only be visible on ⅕ of the floor.

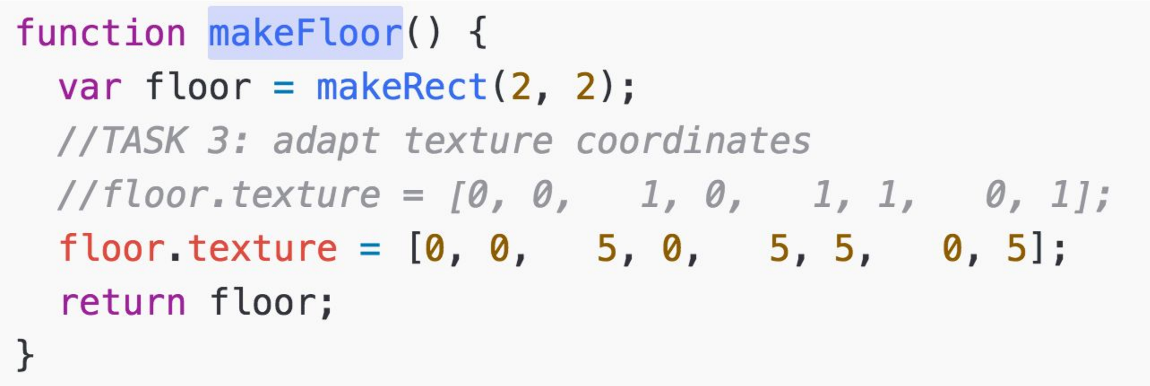
(Any corner is fine)

Task:

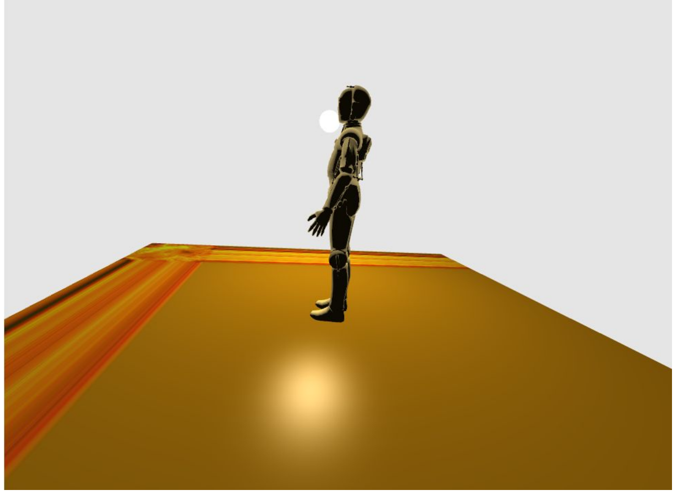
Modify the texture coordinates of the floor to achieve this.



Task 3: main.js Solution



```
function makeFloor() {
  var floor = makeRect(2, 2);
  //TASK 3: adapt texture coordinates
  //floor.texture = [0, 0,   1, 0,   1, 1,   0, 1];
  floor.texture = [0, 0,   5, 0,   5, 5,   0, 5];
  return floor;
}
```



Task 4: Modify Sampling Parameters

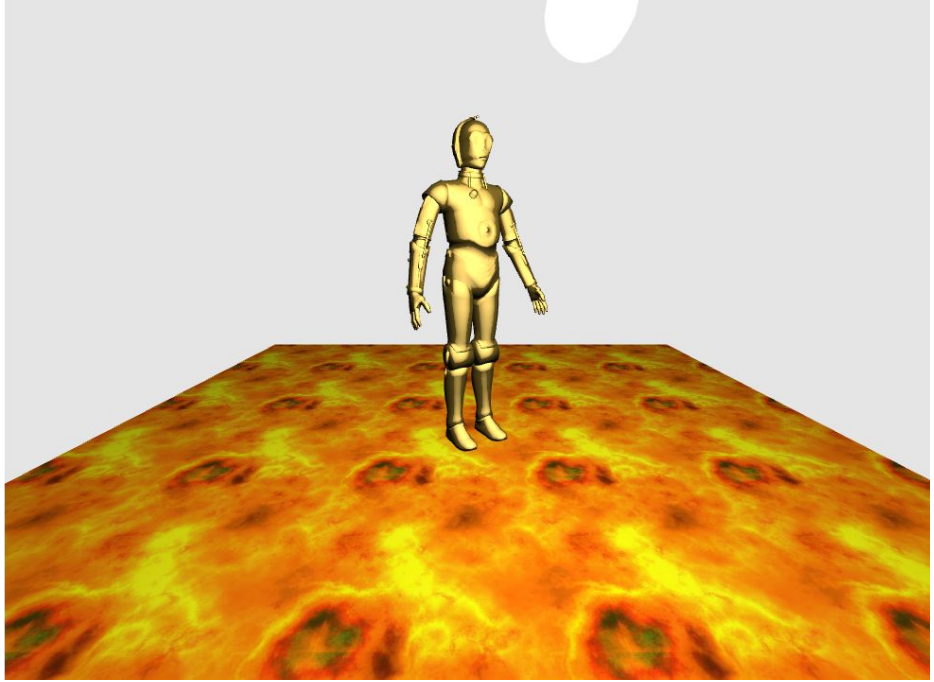
Goal:

The texture should be visible 5x5 times on the floor.

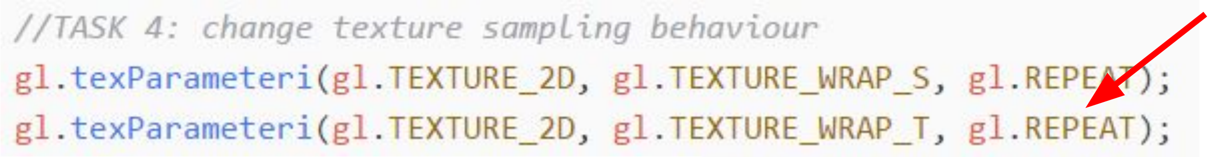
Task:

Modify the texture wrapping parameters to achieve this.

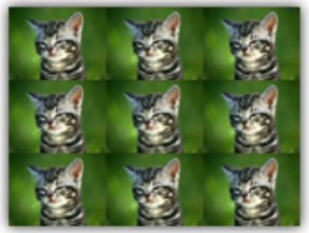
Look up alternative wrapping parameters on the Internet.



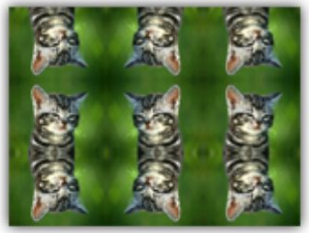
Task 4: main.js Solution



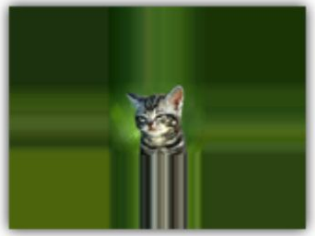
```
//TASK 4: change texture sampling behaviour
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT);
```



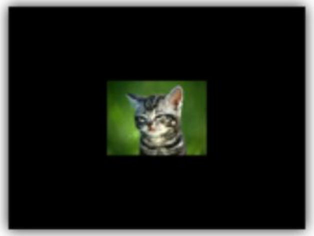
GL_REPEAT



GL_MIRRORED_REPEAT

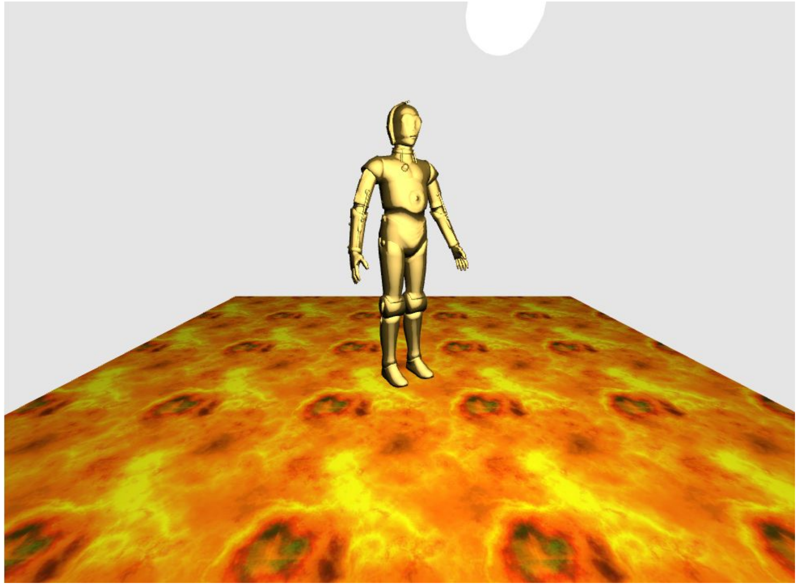


GL_CLAMP_TO_EDGE



GL_CLAMP_TO_BORDER

https://open.gl/textures

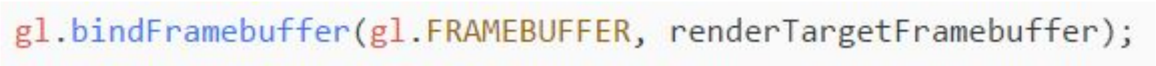


Render to Texture

Multiple render passes

First render pass: render into framebuffer/texture

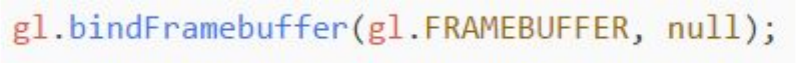
Enable framebuffer:



```
gl.bindFramebuffer(gl.FRAMEBUFFER, renderTargetFramebuffer);
```

Setup viewport + camera + clear buffers + render scene graph

Disable framebuffer:



```
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

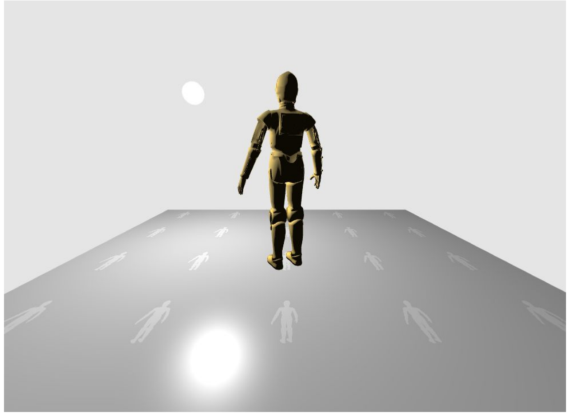
Framebuffer has attached textures to render into.

Nothing will be shown on screen!

Second render pass:

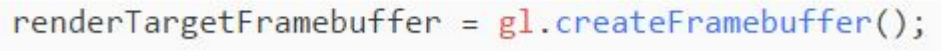
Render scene normally

Texture objects with textures of framebuffer



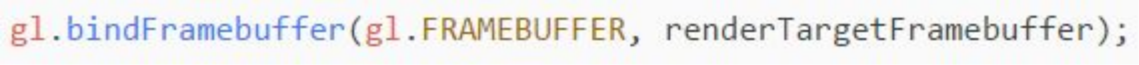
Setup Framebuffer

Create framebuffer object:



```
renderTargetFramebuffer = gl.createFramebuffer();
```

Bind framebuffer to use it (and for setup):



```
gl.bindFramebuffer(gl.FRAMEBUFFER, renderTargetFramebuffer);
```

Create empty textures to render into (setup similar as learned before)

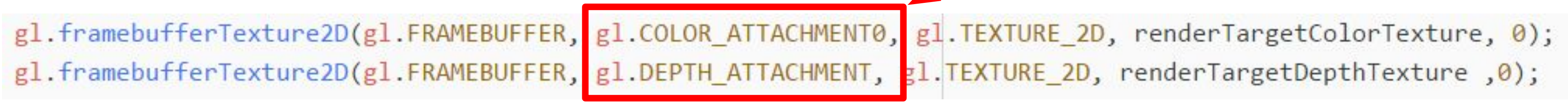
Color texture: format: `gl.RGBA`, image data type: `gl.UNSIGNED_BYTE`

Depth texture: format: `gl.DEPTH_COMPONENT`, image data type: `gl.UNSIGNED_SHORT`

Depth texture required to allow doing depth test!

Attach textures to framebuffer

specify texture usage

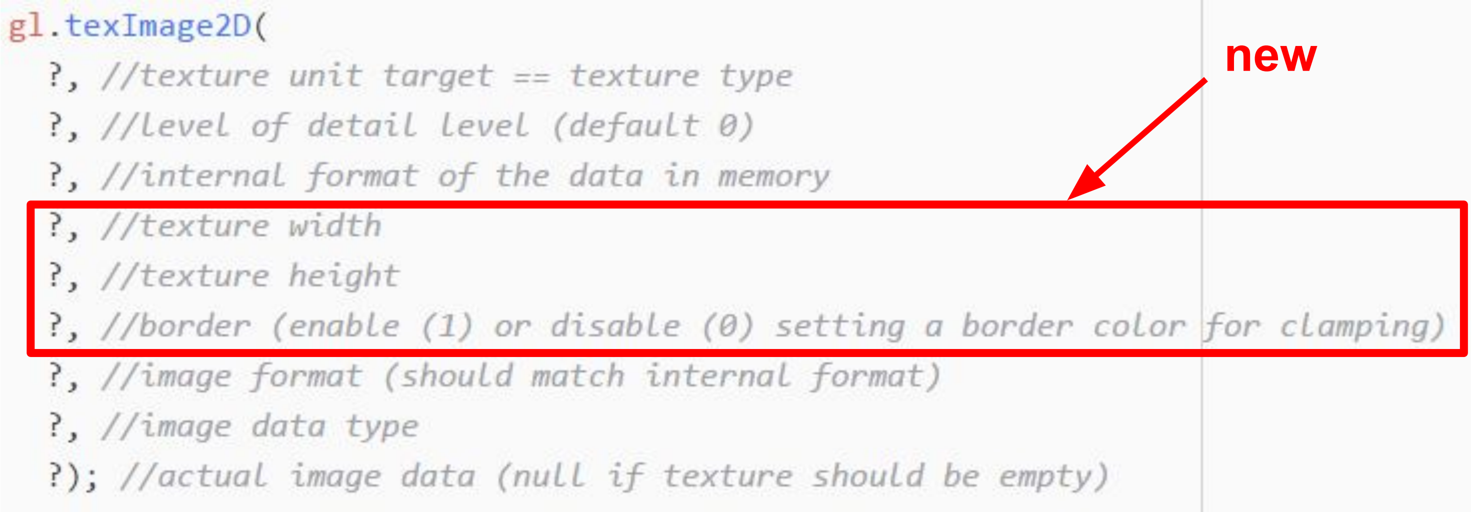


```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, renderTargetColorTexture, 0);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.TEXTURE_2D, renderTargetDepthTexture ,0);
```

Unbind framebuffer

Setup Empty Textures

Same as for previous tasks but extended texImage2D:



```
gl.texImage2D(
  ?, //texture unit target == texture type
  ?, //level of detail level (default 0)
  ?, //internal format of the data in memory
  ?, //texture width
  ?, //texture height
  ?, //border (enable (1) or disable (0) setting a border color for clamping)
  ?, //image format (should match internal format)
  ?, //image data type
  ?); //actual image data (null if texture should be empty)
```

new

Texture width and height define framebuffer size

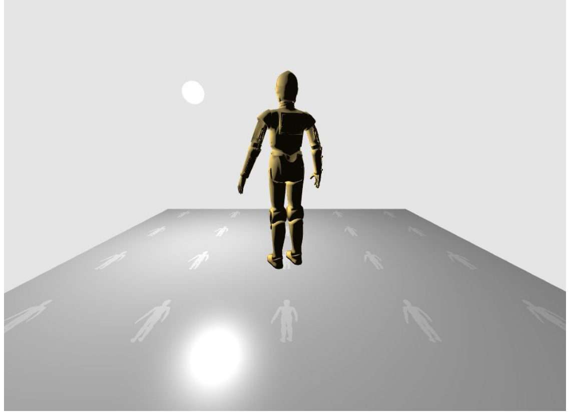
Border can be disabled in our case (0)

Image data is empty (null)

Task 5: Render to Texture

Goal:

Render C3PO into texture and put result on floor.



Given:

Partial framebuffer initialization (`initRenderToTexture`)

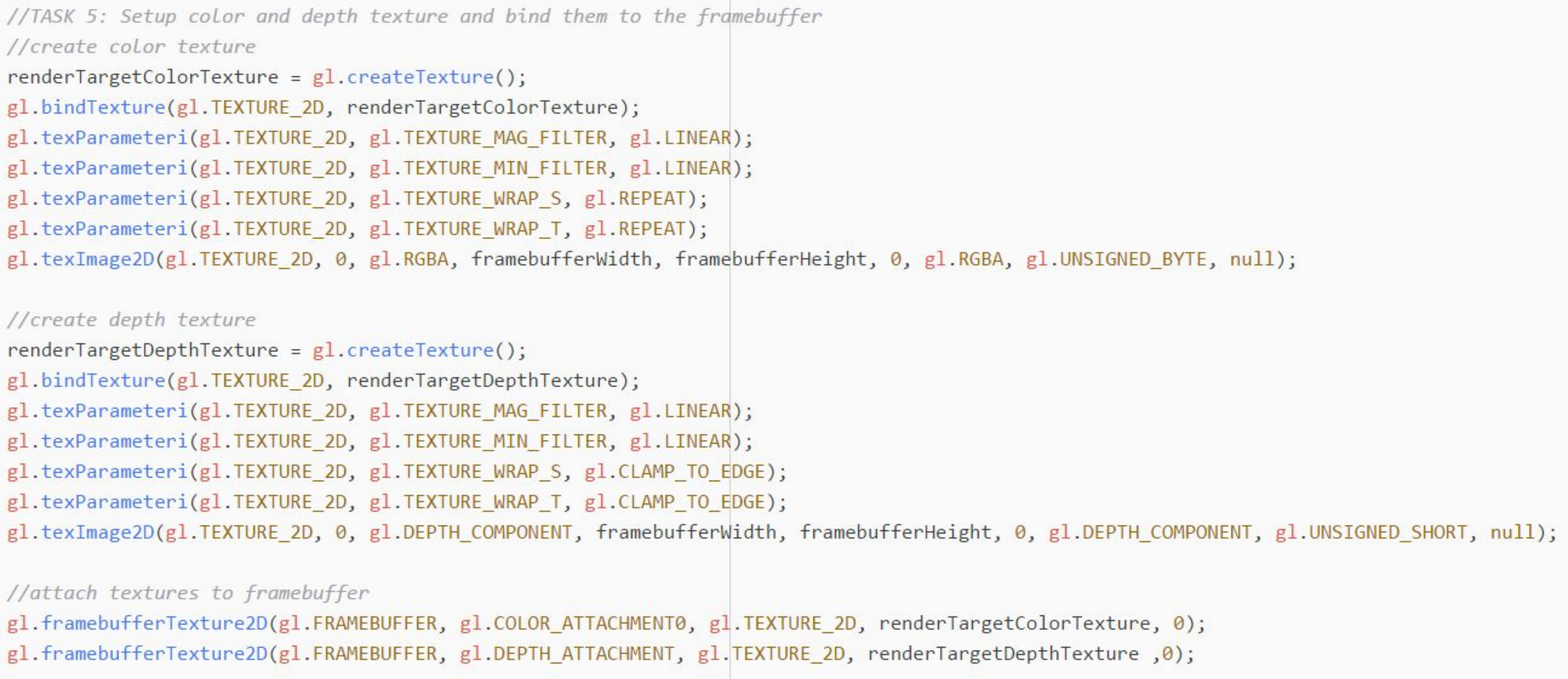
Scene graph without floor and simple shader (`rootnofloor`)

Tasks:

- Initialize textures for color and depth (`initRenderToTexture`)
  - Use `framebufferWidth`, `framebufferHeight` and texture types as in comments
  - Attach those textures to framebuffer and call `initRenderToTexture` function
- Render scene without floor to the texture/framebuffer (`renderToTexture`)
  - Setup projection and view matrix as in normal scene but without mouse rotation
- Put color texture of framebuffer on the floor (`createSceneGraph`)

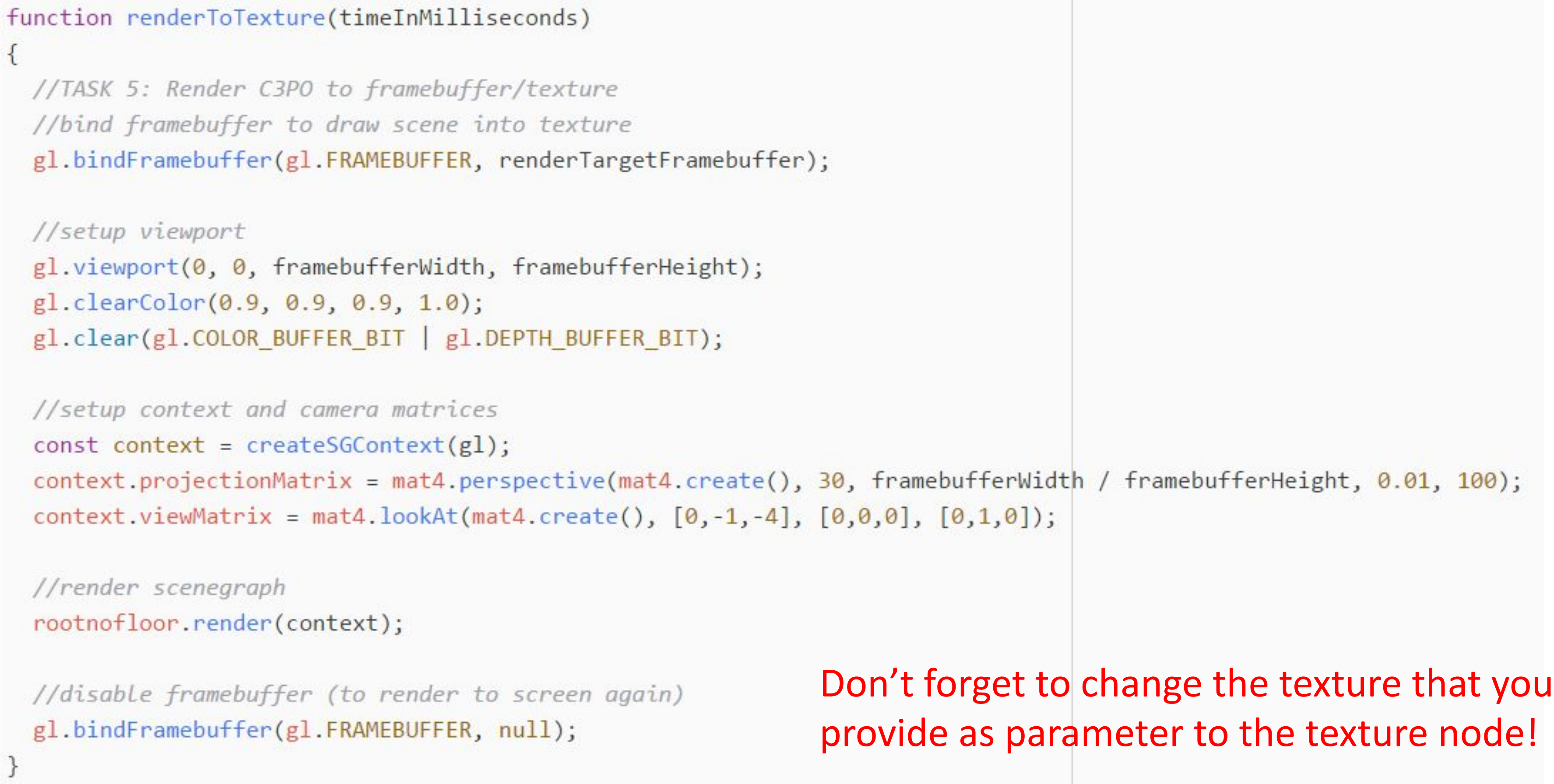
Task 5: main.js Solution

initRenderToTexture function



```
//TASK 5: Setup color and depth texture and bind them to the framebuffer
//create color texture
renderTargetColorTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, renderTargetColorTexture);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, framebufferWidth, framebufferHeight, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);

//create depth texture
renderTargetDepthTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, renderTargetDepthTexture);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.DEPTH_COMPONENT, framebufferWidth, framebufferHeight, 0, gl.DEPTH_COMPONENT, gl.UNSIGNED_SHORT, null);

//attach textures to framebuffer
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, renderTargetColorTexture, 0);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.TEXTURE_2D, renderTargetDepthTexture ,0);
```

Task 5: main.js Solution



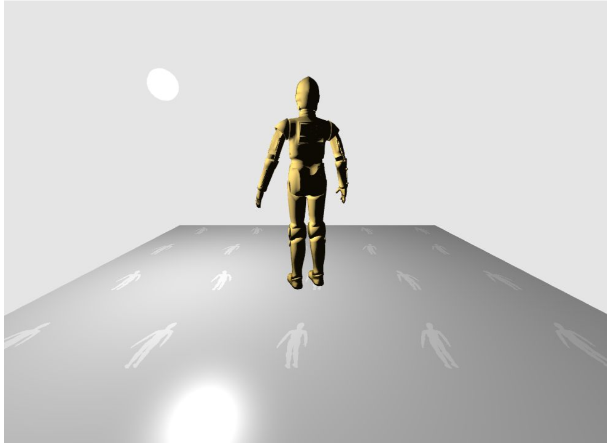
```
function renderToTexture(timeInMilliseconds)
{
  //TASK 5: Render C3PO to framebuffer/texture
  //bind framebuffer to draw scene into texture
  gl.bindFramebuffer(gl.FRAMEBUFFER, renderTargetFramebuffer);

  //setup viewport
  gl.viewport(0, 0, framebufferWidth, framebufferHeight);
  gl.clearColor(0.9, 0.9, 0.9, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  //setup context and camera matrices
  const context = createSGContext(gl);
  context.projectionMatrix = mat4.perspective(mat4.create(), 30, framebufferWidth / framebufferHeight, 0.01, 100);
  context.viewMatrix = mat4.lookAt(mat4.create(), [0,-1,-4], [0,0,0], [0,1,0]);

  //render scenegraph
  rootnofloor.render(context);

  //disable framebuffer (to render to screen again)
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Don’t forget to change the texture that you provide as parameter to the texture node!

Extra Task: Animate Texture Coordinates

Goal:

Achieve wobble effect by animating the texture coordinates in the shader.



No task numbers! Think by yourself ;-)

Hints:

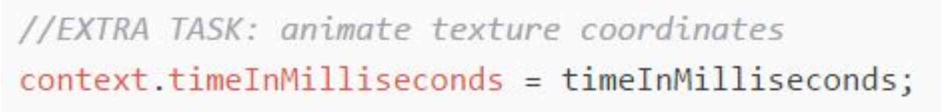
Pass time to shader

Use `sin()` function (e.g. x = x+sin(y+time))

Copy texture coordinates to local variable before modification

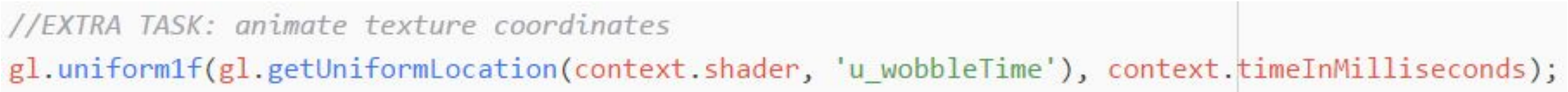
Extra Task: Solution

main.js render functions:



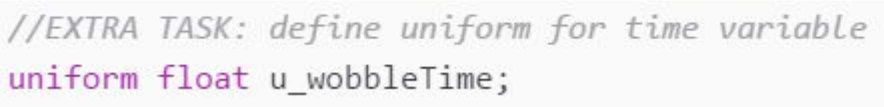
```
//EXTRA TASK: animate texture coordinates
context.timeInMilliseconds = timeInMilliseconds;
```

main.js TextureSGNode:



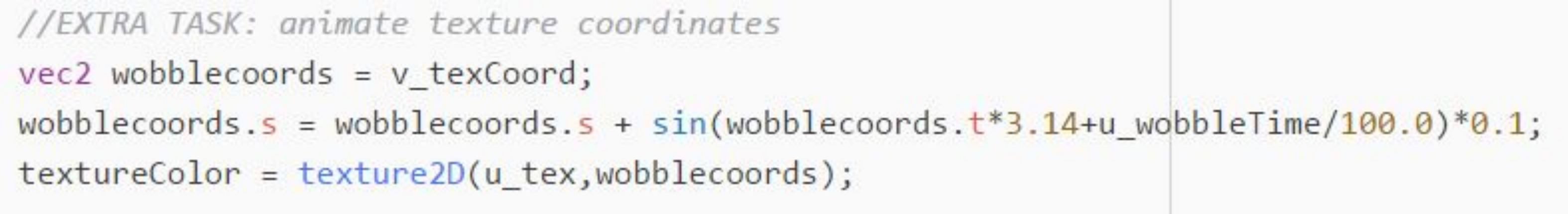
```
//EXTRA TASK: animate texture coordinates
gl.uniform1f(gl.getUniformLocation(context.shader, 'u_wobbleTime'), context.timeInMilliseconds);
```

fragment shader:



```
//EXTRA TASK: define uniform for time variable
uniform float u_wobbleTime;
```

fragment shader main function:



```
//EXTRA TASK: animate texture coordinates
vec2 wobblecoords = v_texCoord;
wobblecoords.s = wobblecoords.s + sin(wobblecoords.t*3.14+u_wobbleTime/100.0)*0.1;
textureColor = texture2D(u_tex,wobblecoords);
```

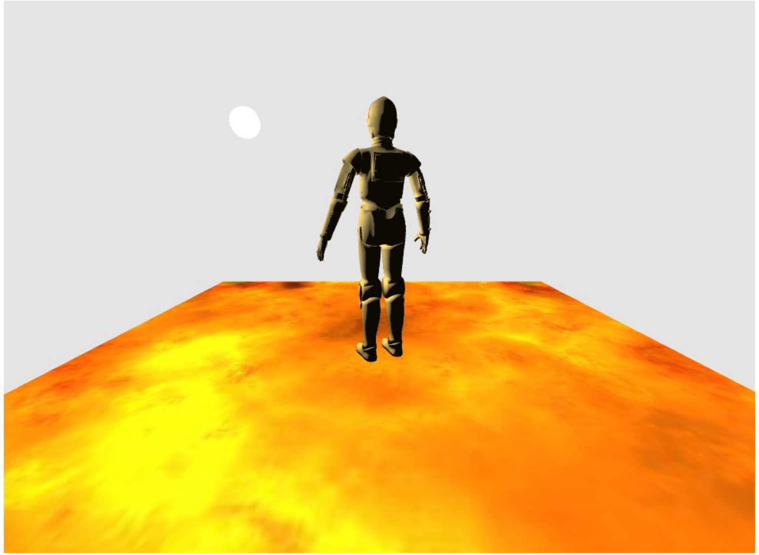
Recap

Texturing Basics

- Setup Textures
- Texture Coordinates
- Textures in Shader
- Binding Textures
- Integrate Texturing into Phong Shader
- Texture Wrapping and Repeating

Rendering to Textures

- Framebuffer
- Multiple Render Passes



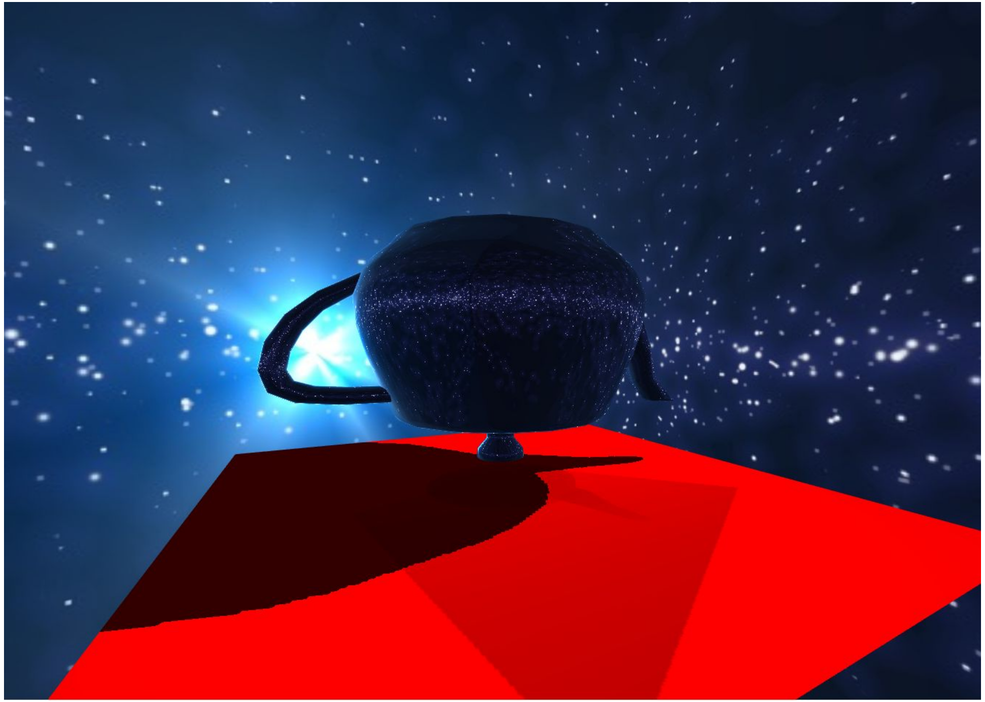
Next Time

JOHANNES KEPLER
UNIVERSITY LINZ

INSTITUTE OF
COMPUTER GRAPHICS

Advanced Texture Mapping

- Environment Mapping
- Shadow Mapping



Thanks!

Have fun with your CG-Projects.

Questions / Feedback: cg-lab@jku.at