JOHANNES KEPLER
UNIVERSITY LINZ
INSTITUTE OF
COMPUTER GRAPHICS

Computer Graphics

Lab 4: Illumination and Shading

Dev Environment: Lab Package

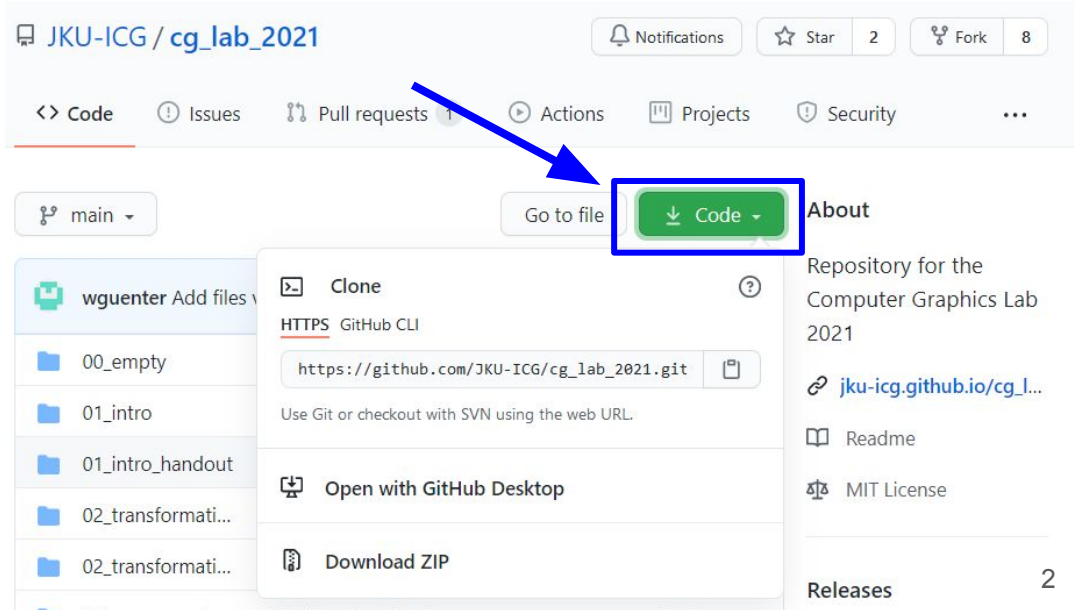
JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

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

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

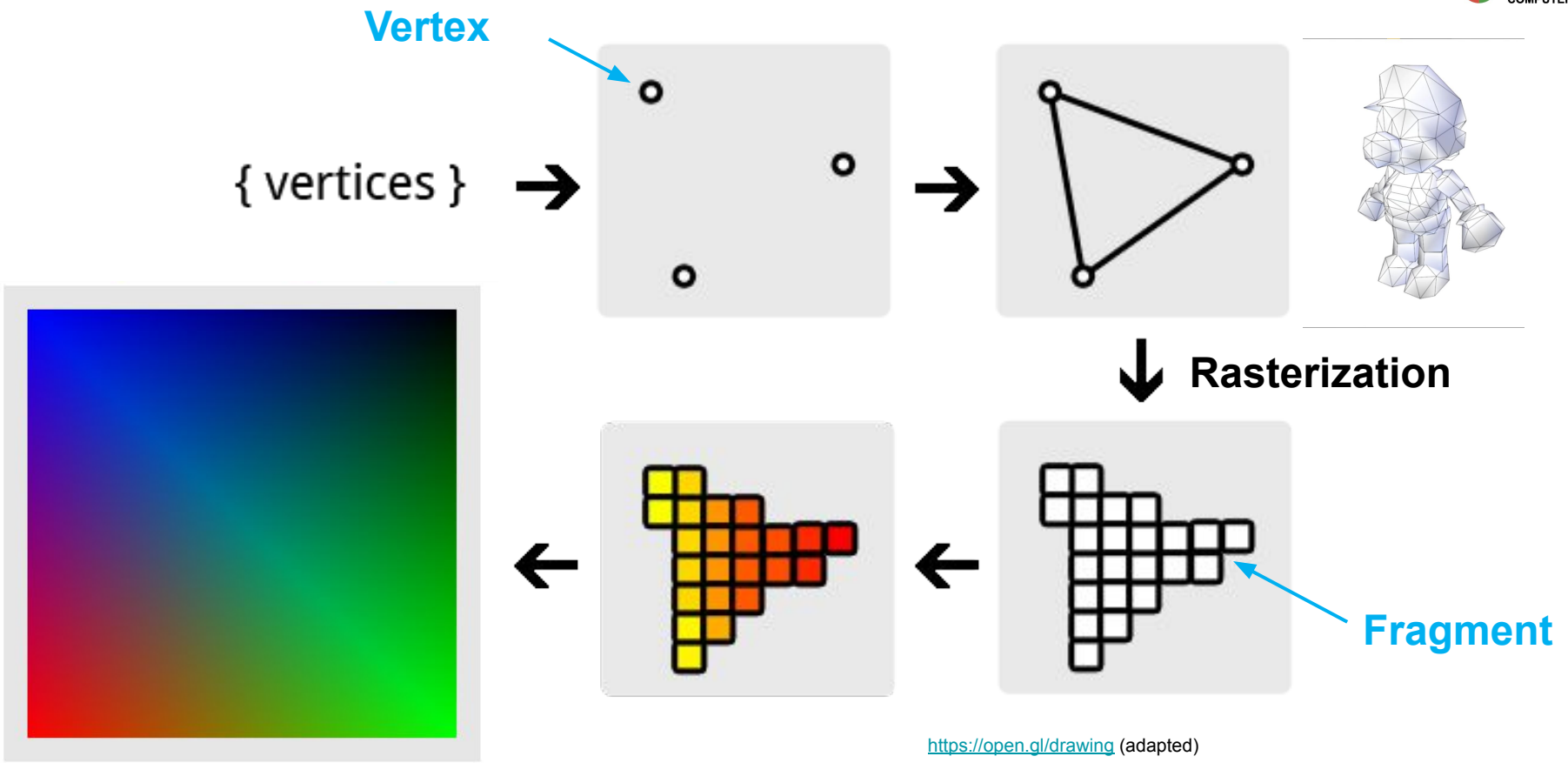
To get started (now):

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

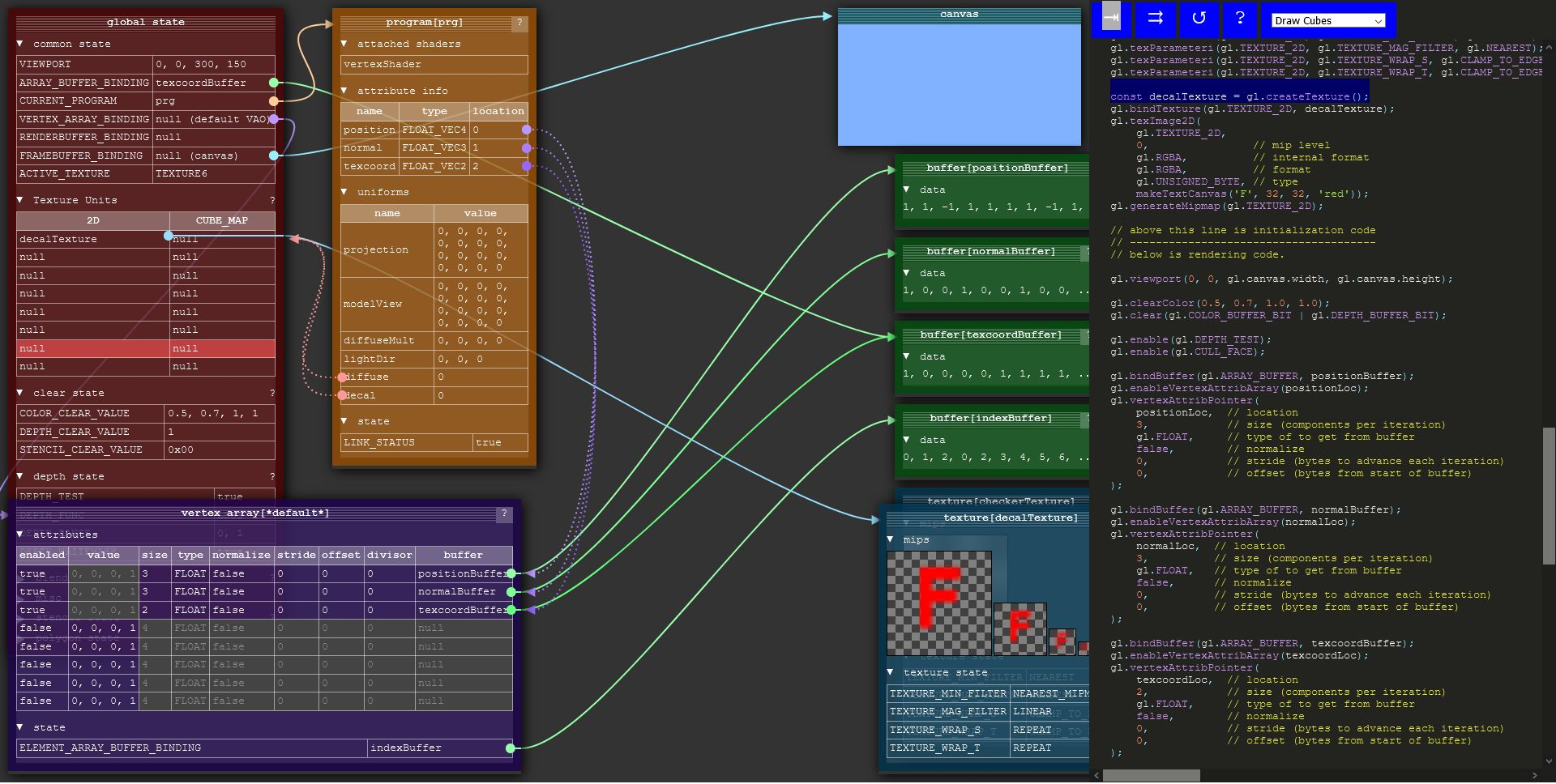


Rendering Pipeline

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

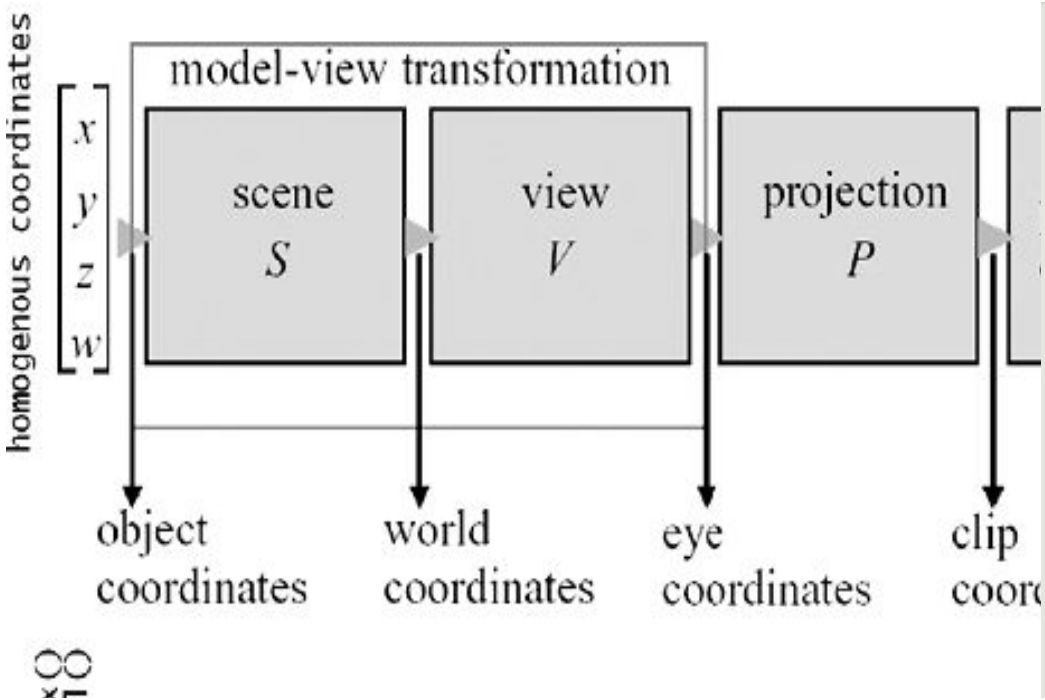


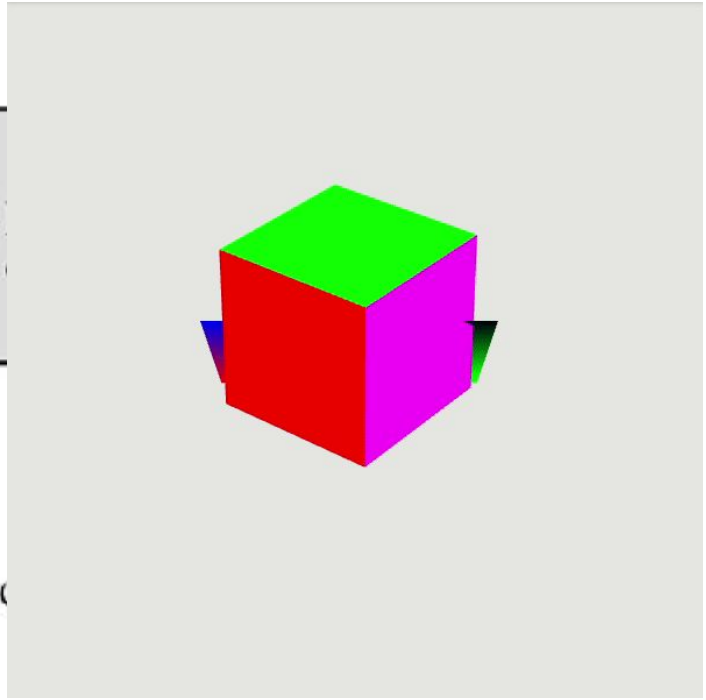
https://open.gl/drawing (adapted)



WebGL State Diagram: https://webglfundamentals.org/webgl/lessons/resources/webgl-state-diagram.html

Transformation Pipeline

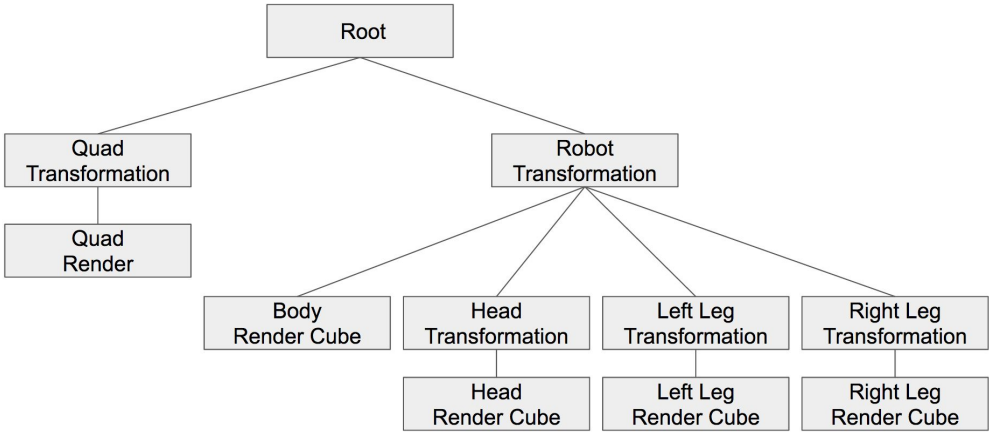


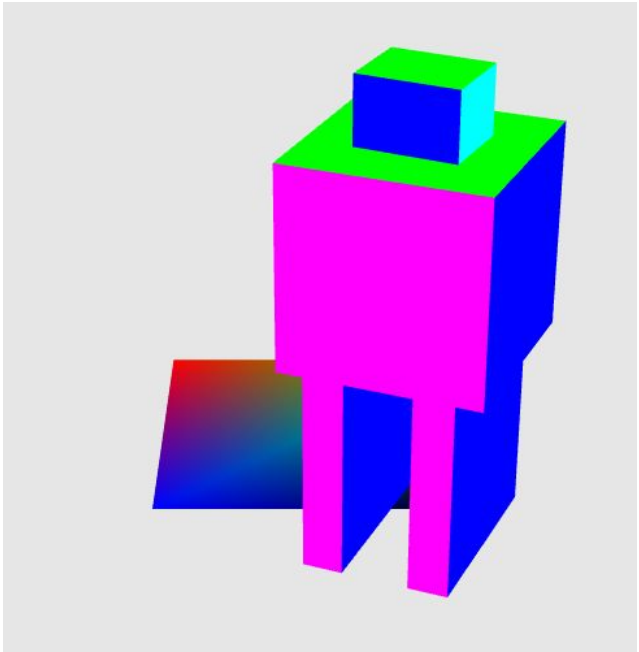


Recap: Lab 3

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

- Depth Handling
- Blending
- Scene Graphs
  - Abstraction into nodes
  - Scene graph traversal
  - Implemented robot using a scene graph



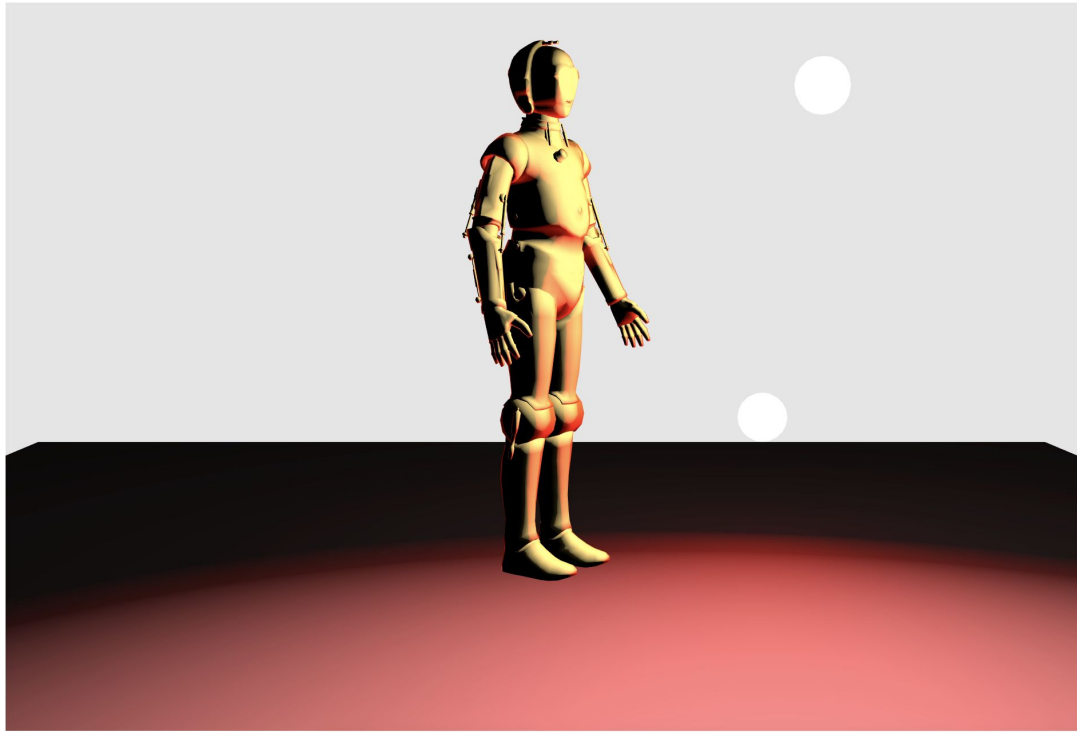


Agenda for Today

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Illumination

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

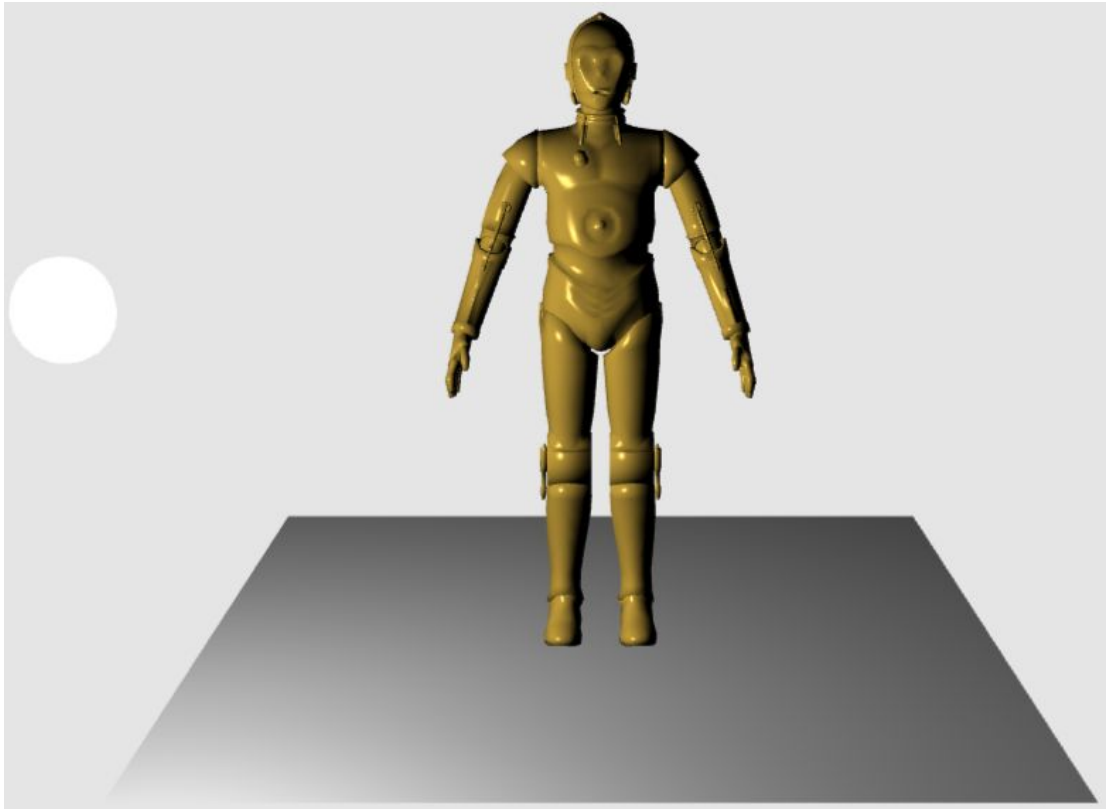


Why do we need illumination?

Not really 3D, right?
Looks flat and boring

Maybe, if we rotate and interact with the scene...

Let there be light



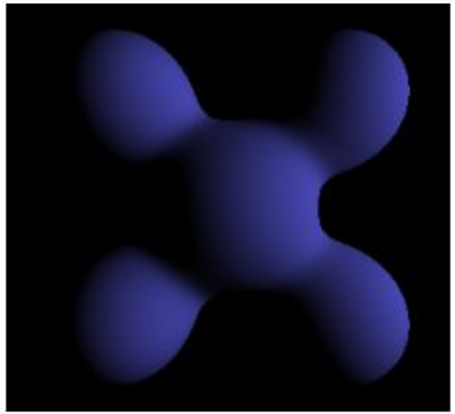
Reveals shape

Shows details

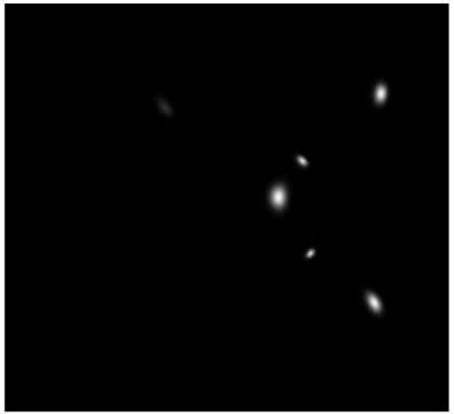
Humans are used to shaded objects.

Illumination Model: Phong Shading

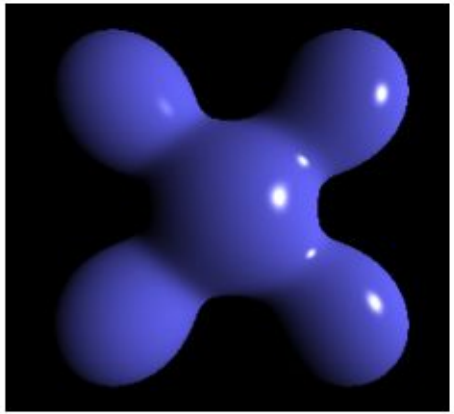
ambient / emissive



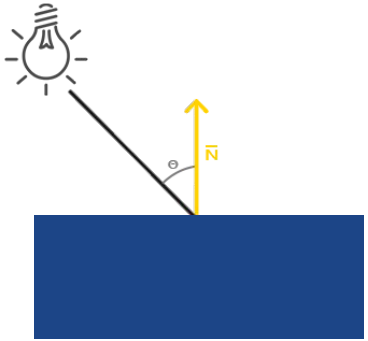
+ diffuse

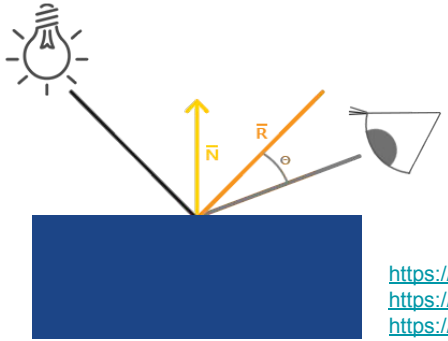


+ specular



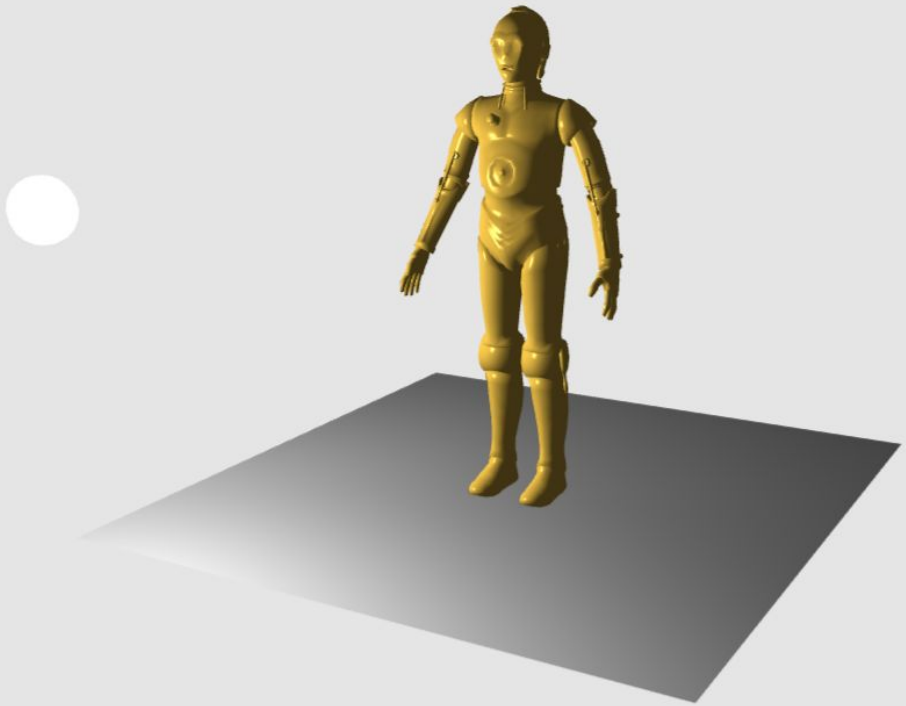
= Phong





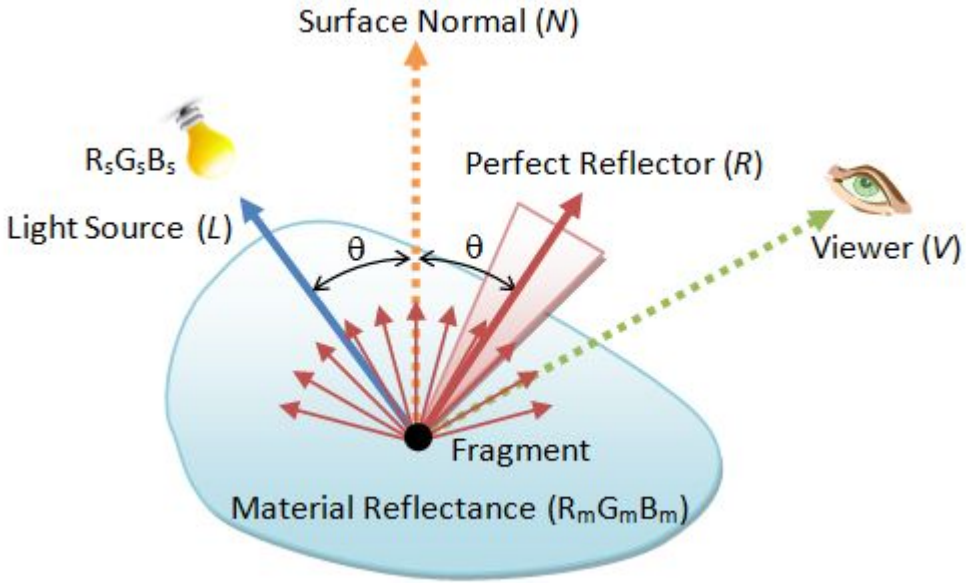
https://learnopengl.com/Lighting/Basic-Lighting
https://en.wikipedia.org/wiki/Phong_reflection_model
https://youtu.be/RjA_sC4bCAM

Interactive Demo



https://jku-icg.github.io/cg_demo/00_shading/

Phong Shading



$$c_{amb} = l_{amb} * m_{amb}$$

$$c_{diff} = max(L \cdot N, 0) * l_{diff} * m_{diff}$$

$$c_{spec} = max(R \cdot V, 0)^{m\_sh} * l_{spec} * m_{spec}$$

$$c_{em} = m_{em}$$

$$\boldsymbol{c_{final}} = c_{amb} + c_{diff} + c_{spec} + c_{em}$$

c = color

l = light

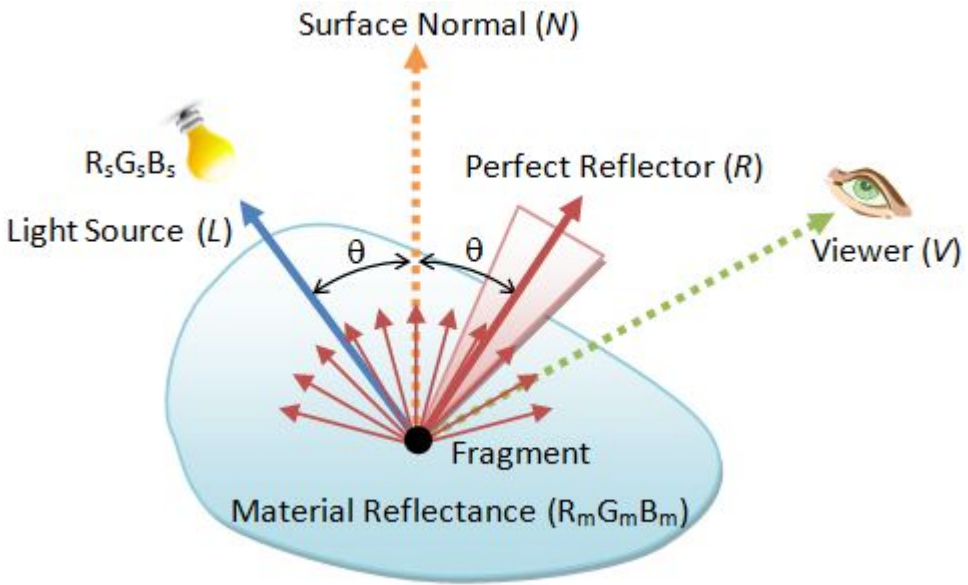
amb = ambient, diff = diffuse, spec = specular, em = emission

m = material, m_sh = material shininess

N = normal vector, R = reflection vector, V = view vector

http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Task 1: Implement Phong Shader



$$c_{amb} = l_{amb} * m_{amb}$$

$$c_{diff} = max(L \cdot N, 0) * l_{diff} * m_{diff}$$

$$c_{spec} = max(R \cdot V, 0)^{m\_sh} * l_{spec} * m_{spec}$$

$$c_{em} = m_{em}$$

$$\boldsymbol{c_{final}} = c_{amb} + c_{diff} + c_{spec} + c_{em}$$

c = color

l = light

amb = ambient, diff = diffuse, spec = specular, em = emission

m = material, m_sh = material shininess

N = normal vector, R = reflection vector, V = view vector

http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Task 1: Implement Phong Shader

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Some useful math functions

`min(x,y)`	Returns y if y < x, otherwise it returns x.
`max(x,y)`	Returns y if x < y, otherwise it returns x.
`clamp(x,min,max)`	Returns x, if min < x < max, otherwise min/max.
`pow(x,y)`	Returns x raised to the y power, i.e., x^y.
`dot(x,y)`	Returns the dot product of x and y.
`reflect(I,N)`	For the **incident** vector I and surface orientation N, returns the reflection direction. N must already be normalized in order to achieve the desired result.

Functions provided by the shading language:

- https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.1.10.pdf (Chapter 8)

Phong Shader

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Vertex Shader (*phong.vs.glsl*):

```
12  //light position
13  vec3 lightPos = vec3(0.0, -2.0, 2.0);
14
15  //output of this shader
16  varying vec3 v_normalVec;
17  varying vec3 v_eyeVec;
18  varying vec3 v_lightVec;
19
20  void main() {
21    vec4 eyePosition = u_modelView * vec4(a_position,1);
22
23    v_normalVec = u_normalMatrix * a_normal;
24
25    v_eyeVec = -eyePosition.xyz;
26
27    v_lightVec = lightPos - eyePosition.xyz;
28
29    gl_Position = u_projection * eyePosition;
30  }
31
```

Fragment Shader (*phong.fs.glsl*):

```
46  vec4 calculateSimplePointLight(Light light, Material material, vec3 lightVec,
47                                 vec3 normalVec, vec3 eyeVec) {
48    lightVec = normalize(lightVec);
49    normalVec = normalize(normalVec);
50    eyeVec = normalize(eyeVec);
51
52    //TASK 1-1 implement phong shader
53    //compute diffuse term
54    float diffuse = 0.0;
55
56    //compute specular term
57    vec3 reflectVec = vec3(0.0, 0.0, 0.0);
58    float spec = 0.0;
59
60    //use term an light to compute the components
61    vec4 c_amb  = clamp(material.ambient, 0.0, 1.0);
62    vec4 c_diff = clamp(material.diffuse, 0.0, 1.0);
63    vec4 c_spec = clamp(material.specular, 0.0, 1.0);
64    vec4 c_em   = material.emission;
65
66    return c_amb + c_diff + c_spec + c_em;
67  }
68
69  void main() {
70    //Task 2-3 use material uniform
71    //Task 3-3 use light uniform
72    //Task 4-3 use second light source
73    gl_FragColor = calculateSimplePointLight(light, material, v_lightVec,
74                                             v_normalVec, v_eyeVec);
75
```

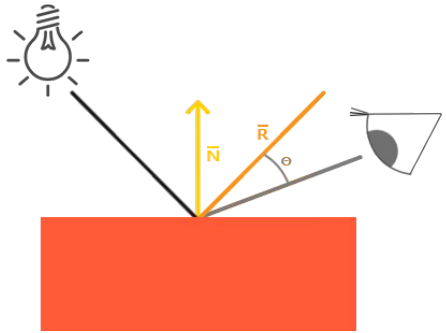
Phong Shader

Vertex Shader (*phong.vs.glsl*):

```
12    //light position
13    vec3 lightPos = vec3(0.0, -2.0, 2.0);
14
15    //output of this shader
16    varying vec3 v_normalVec;
17    varying vec3 v_eyeVec;
18    varying vec3 v_lightVec;
19
20    void main() {
21      vec4 eyePosition = u_modelView * vec4(a_position,1);
22
23      v_normalVec = u_normalMatrix * a_normal;
24
25      v_eyeVec = -eyePosition.xyz;
26
27      v_lightVec = lightPos - eyePosition.xyz;
28
29      gl_Position = u_projection * eyePosition;
30    }
31
```

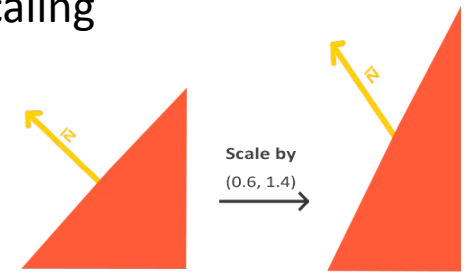
Computation Requires:

- Light position
- Surface normals
- View position (origin)



Normal Matrix

3x3 without scaling



https://learnopengl.com/Lighting/Basic-Lighting
http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/the-normal-matrix/

Light

Specified by:

- ambient (color)
- diffuse (color)
- specular (color)
- position (coordinate)
- others (e.g., angle)

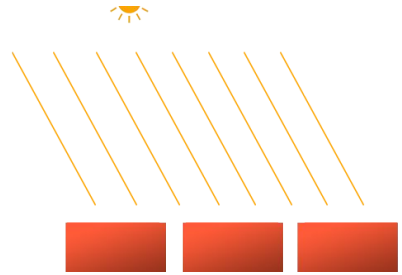
Fragment Shader (*phong.fs.glsl*):

```
struct Light {
  vec4 ambient;
  vec4 diffuse;
  vec4 specular;
};
```

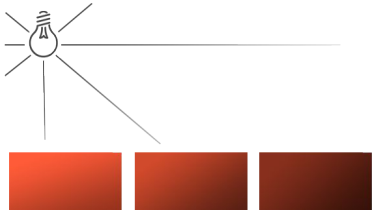
```
Light light = Light(vec4(0., 0., 0., 1.),
                    vec4(1., 1., 1., 1.),
                    vec4(1., 1., 1., 1.));
```

Types:

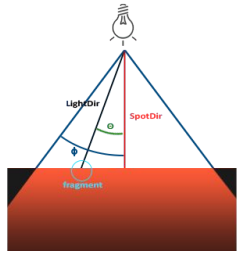
directional



point



spotlight



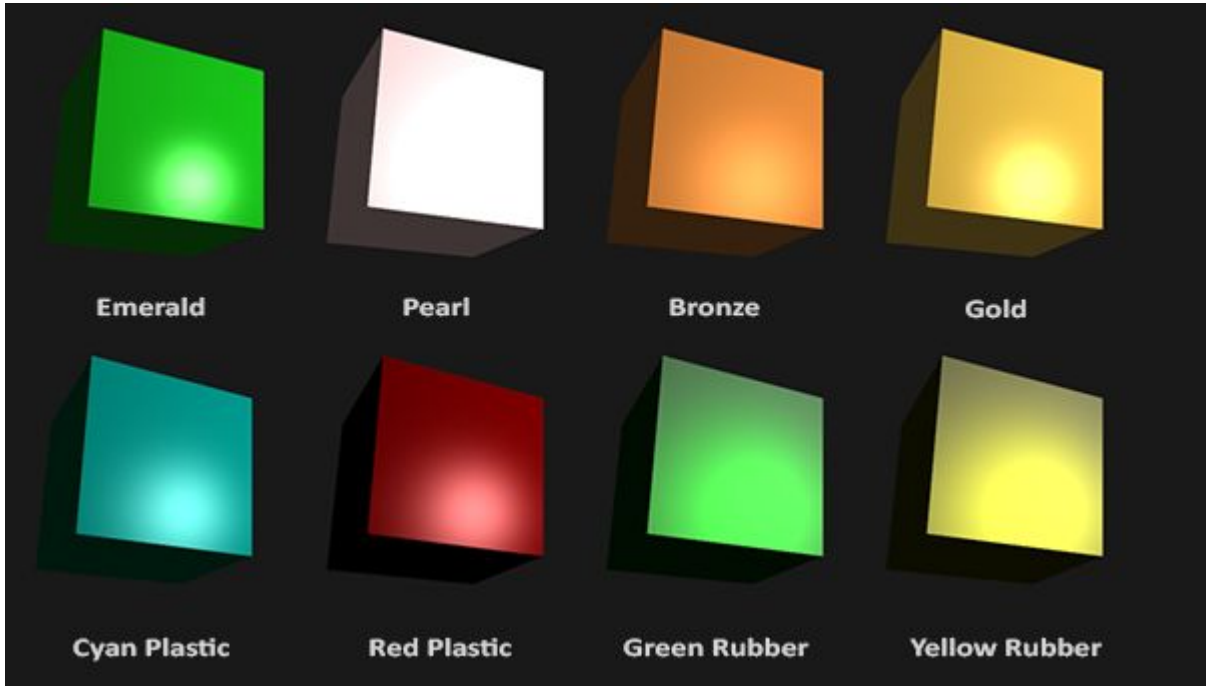
https://learnopengl.com/Lighting/Light-casters

Materials

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Specified by:

- ambient (color)
- diffuse (color)
- specular (color)
- emission (color)
- shininess (scalar)



Fragment Shader (*phong.fs.glsl)*:

```
struct Material {
  vec4 ambient;
  vec4 diffuse;
  vec4 specular;
  vec4 emission;
  float shininess;
};
```

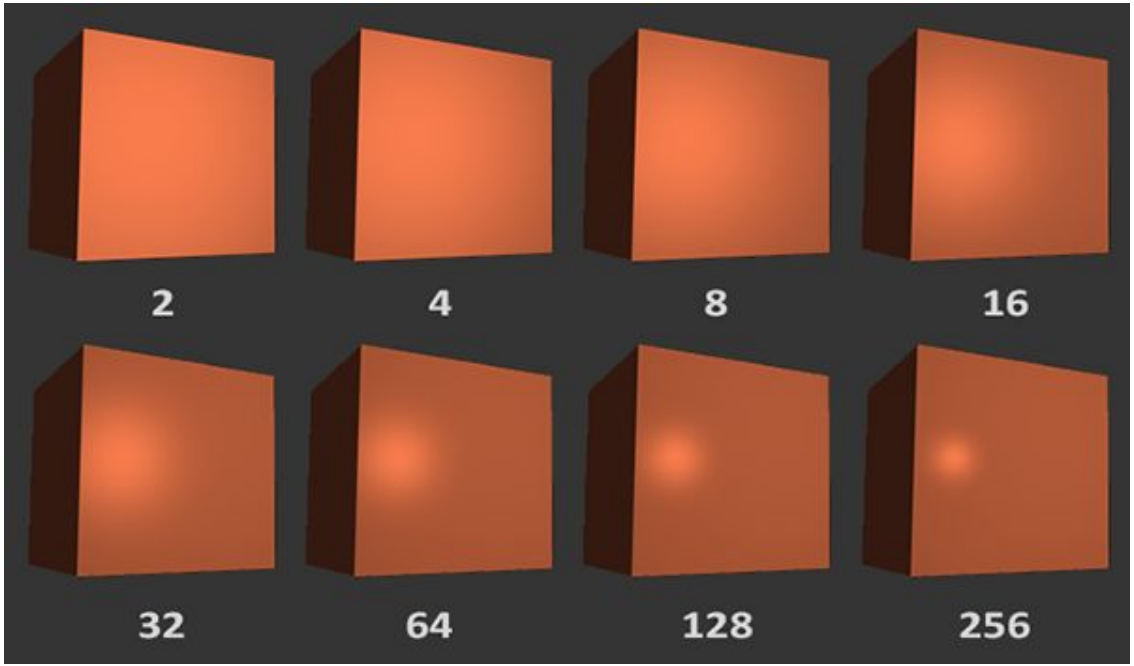
```
Material material = Material(vec4(0.24725, 0.1995, 0.0745, 1.),
                             vec4(0.75164, 0.60648, 0.22648, 1.),
                             vec4(0.628281, 0.555802, 0.366065, 1.),
                             vec4(0., 0., 0., 0.),
                             0.4);
```

https://learnopengl.com/Lighting/Materials

Material: Impact of Shininess

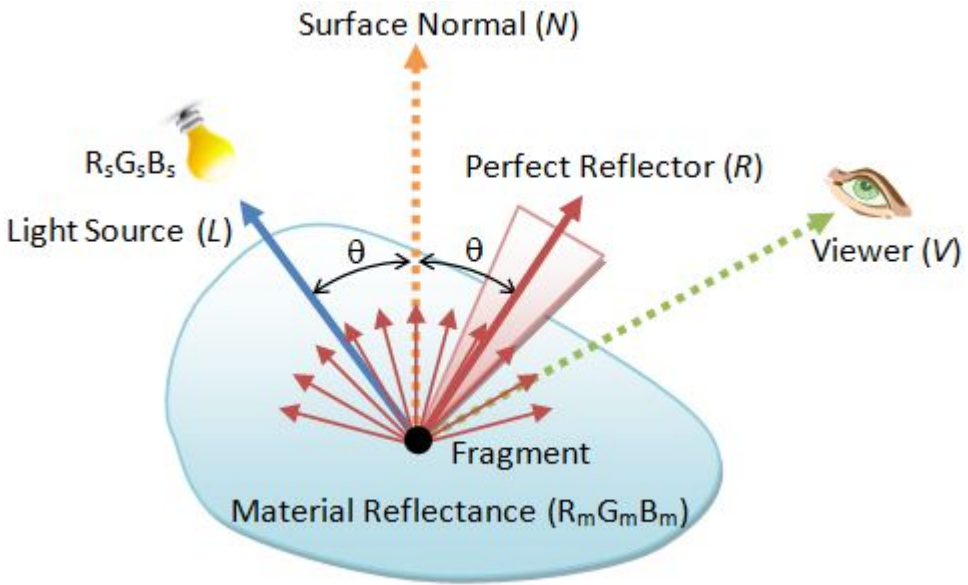
JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

High shininess (exponent) causes small highlight



Test it yourself: https://jku-icg.github.io/cg_demo/00_shading/

Task 1: Implement Phong Shader



$$c_{amb} = l_{amb} * m_{amb}$$

$$c_{diff} = max(L \cdot N, 0) * l_{diff} * m_{diff}$$

$$c_{spec} = max(R \cdot V, 0)^{m\_sh} * l_{spec} * m_{spec}$$

$$c_{em} = m_{em}$$

$$\boldsymbol{c_{final}} = c_{amb} + c_{diff} + c_{spec} + c_{em}$$

c = color

l = light

amb = ambient, diff = diffuse, spec = specular, em = emission

m = material, m_sh = material shininess

N = normal vector, R = reflection vector, V = view vector

http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Task 1: Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```glsl
vec4 calculateSimplePointLight(Light light, Material material, vec3 lightVec,
                               vec3 normalVec, vec3 eyeVec) {
  lightVec = normalize(lightVec);
  normalVec = normalize(normalVec);
  eyeVec = normalize(eyeVec);

  //compute diffuse term
  float diffuse = max(dot(normalVec,lightVec),0.0);

  //compute specular term
  vec3 reflectVec = reflect(-lightVec,normalVec);
  float spec = pow( max( dot(reflectVec, eyeVec), 0.0) , material.shininess);


  vec4 c_amb  = clamp(light.ambient * material.ambient, 0.0, 1.0);
  vec4 c_diff = clamp(diffuse * light.diffuse * material.diffuse, 0.0, 1.0);
  vec4 c_spec = clamp(spec * light.specular * material.specular, 0.0, 1.0);
  vec4 c_em   = material.emission;

  return c_amb + c_diff + c_spec + c_em;
}
```

phong.fs.glsl

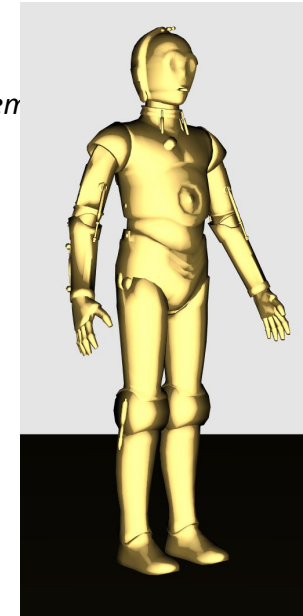
$$c_{amb} = l_{amb} * m_{amb}$$

$$c_{diff} = \boxed{max(L \cdot N, 0)} * l_{diff} * m_{diff}$$

$$c_{spec} = \boxed{max(R \cdot V, 0)^{m\_sh}} * l_{spec} * m_{spec}$$

$$c_{em} = m_{em}$$

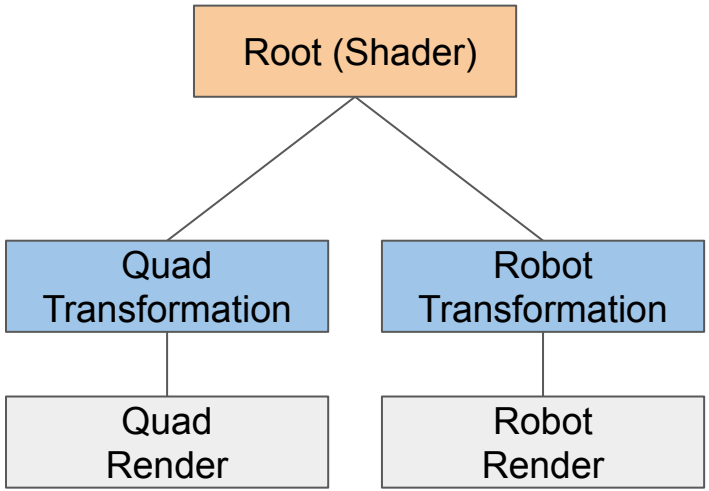
$$\boldsymbol{c_{final}} = c_{amb} + c_{diff} + c_{spec} + c_{em}$$



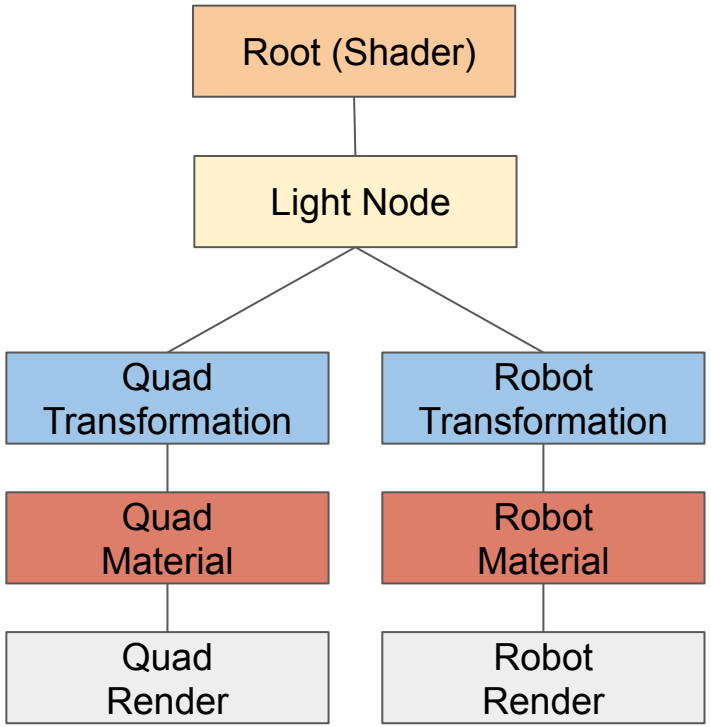
Scene Graph

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Current Scene Graph:



Extract Hardcoded Light & Material:



Task 2: Extract to MaterialNode

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```
10  struct Material {
11    vec4 ambient;
12    vec4 diffuse;
13    vec4 specular;
14    vec4 emission;
15    float shininess;
16  };
17
18  /**
19   * definition of the light properties related to material properties
20   */
21  struct Light {
22    vec4 ambient;
23    vec4 diffuse;
24    vec4 specular;
25  };
26
27  //TASK 2-1 use uniform for material
28  Material material = Material(vec4(0.24725, 0.1995, 0.0745, 1.),
29                               vec4(0.75164, 0.60648, 0.22648, 1.),
30                               vec4(0.628281, 0.555802, 0.366065, 1.),
31                               vec4(0., 0., 0., 0.),
32                               0.4);
33
34  //TASK 3-1 use uniform for light
35  Light light = Light(vec4(0., 0., 0., 1.),
36                      vec4(1., 1., 1., 1.),
37                      vec4(1., 1., 1., 1.));
```

phong.fs.glsl

Hard coded material and light properties are not optimal.

- 2-1, 2-2 fragment shader: define uniform `u_material` and use it (instead of `material` defined in shader)
- 2-3 main.js: finish `MaterialNode` class by setting the uniforms
- 2-4 main.js: wrap c3p0 with new node and set material to the shader node
- 2-5 main.js: wrap floor with material

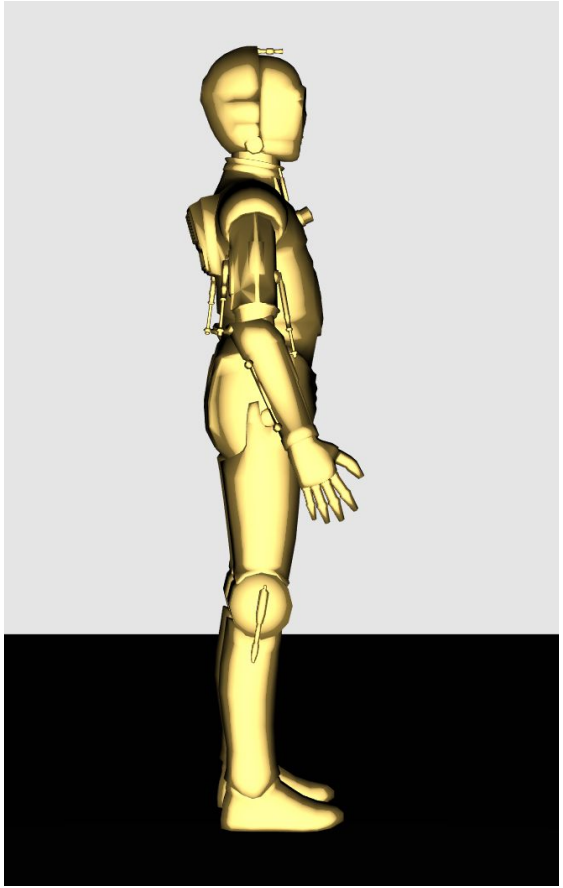
```
floor.ambient = [0, 0, 0, 1];
floor.diffuse = [0.1, 0.1, 0.1, 1];
floor.specular = [0.5, 0.5, 0.5, 1];
floor.emission = [0, 0, 0, 1];
```

Task 2: Fragment Shader Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```
27  //TASK 2-1 use uniform for material
28  //Material material = Material(vec4(0.24725, 0.1995, 0.0745, 1.),
29  //                             vec4(0.75164, 0.60648, 0.22648, 1.),
30  //                             vec4(0.628281, 0.555802, 0.366065, 1.),
31  //                             vec4(0., 0., 0., 0.),
32  //                             0.4);
33  uniform Material u_material;
```

```
69  void main() {
70    //TASK 2-3 use material uniform
71    //TASK 3-2 use light uniform
72    //TASK 5-6 use second light source
73    gl_FragColor = calculateSimplePointLight(light, u_material, v_lightVec,
74                                             v_normalVec, v_eyeVec);
75
76  }
```



phong.fs.glsl

Task 2: main.js Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

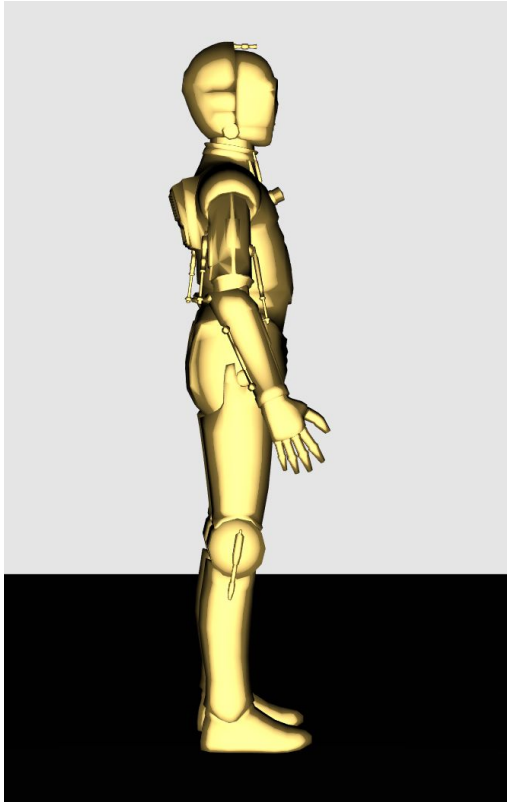
```
setMaterialUniforms(context) {
  const gl = context.gl,
    shader = context.shader;

  //TASK 2-3 set uniforms
  //hint setting a structure element using the dot notation, e.g. u_material.test
  gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.ambient'), this.ambient);
  gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.diffuse'), this.diffuse);
  gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.specular'), this.specular);
  gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.emission'), this.emission);
  gl.uniform1f(gl.getUniformLocation(shader, this.uniform+'.shininess'), this.shininess);
}
```

```
{
  //TASK 2-4 wrap with material node
  let c3po = new MaterialNode([
    new RenderSGNode(resources.model)
  ]);
  //gold
  c3po.ambient = [0.24725, 0.1995, 0.0745, 1];
  c3po.diffuse = [0.75164, 0.60648, 0.22648, 1];
  c3po.specular = [0.628281, 0.555802, 0.366065, 1];
  c3po.shininess = 0.4;
```

```
{
  //TASK 2-5 wrap with material node
  let floor = new MaterialNode([
    new RenderSGNode(makeRect())
  ]);

  //dark
  floor.ambient = [0, 0, 0, 1];
  floor.diffuse = [0.1, 0.1, 0.1, 1];
  floor.specular = [0.5, 0.5, 0.5, 1];
```



main.js

Task 3: Extract Light Node

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Do the same for the light source → extract to `LightNode`

3-1, 3-2 fragment shader: define uniform *`u_light`* and use it

3-3, 3-4 vertex shader: define uniform `u_lightPos` and use it

3-5 main.js: finish `LightNode` class

3-6 main.js: create a white light node at `[0, 2, 2]`

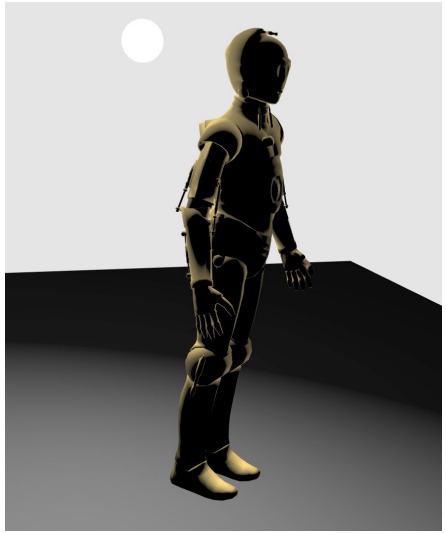
+ append the return value of `createLightSphere()` as a child

Task 3: Shader Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Fragment shader:

```
33   uniform Material u_material;
34   //TASK 3-1 use uniform for light
35   //Light light = Light(vec4(0., 0., 0., 1.),
36   //                    vec4(1., 1., 1., 1.),
37   //                    vec4(1., 1., 1., 1.));
38   uniform Light u_light;
39   //TASK 5-5 use uniform for 2nd light
```



Vertex shader:

```
12   //TASK 3-3 light position as uniform
13   //vec3 lightPos = vec3(0, -2, 2);
14   uniform vec3 u_lightPos;
15   //TASK 5-3 second light source
16
```

```
29     //TASK 3-4 light position as uniform
30     v_lightVec = u_lightPos - eyePosition.xyz;
31     //TASK 5-4 second light source position
32
```

phong.vs.glsl

```
70   void main() {
71     //TASK 2-3 use material uniform
72     //TASK 3-2 use light uniform
73     //TASK 5-6 use second light source
74     gl_FragColor =
75       calculateSimplePointLight(u_light, u_material, v_lightVec, v_normalVec, v_eyeVec);
76
77   }
```

phong.fs.glsl

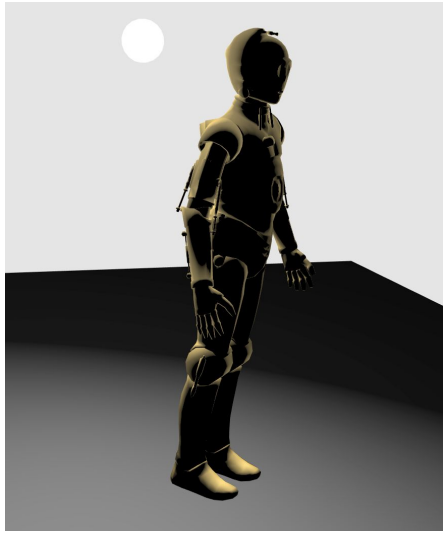
Task 3: main.js Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```
196     setLightUniforms(context) {
197       const gl = context.gl,
198         shader = context.shader,
199         position = this.computeLightPosition(context);
200
201       //TASK 3-5 set uniforms
202       gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.ambient'), this.ambient);
203       gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.diffuse'), this.diffuse);
204       gl.uniform4fv(gl.getUniformLocation(shader, this.uniform+'.specular'), this.specular);
205
206       gl.uniform3f(gl.getUniformLocation(shader, this.uniform+'Pos'), position[0], position[1], position[2]);
207     }
```

```
//TASK 3-6 create white light node at [0, 2, 2]
let light = new LightNode();
light.ambient = [0, 0, 0, 1];
light.diffuse = [1, 1, 1, 1];
light.specular = [1, 1, 1, 1];
light.position = [0, 2, 2];
light.append(createLightSphere());
root.append(light);
```

main.js



Task 4: Animate Light Source

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Static lights are boring. Let’s animate the light source.

4-1 main.js: wrap the light node with a transformation node and store the transformation node in `rotateLight` (which is already defined)

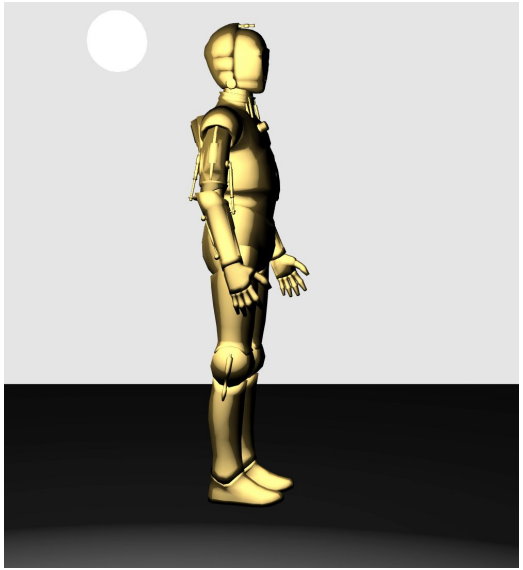
4-2 main.js: apply an animation of `rotateLight` in the render method by setting the rotation matrix

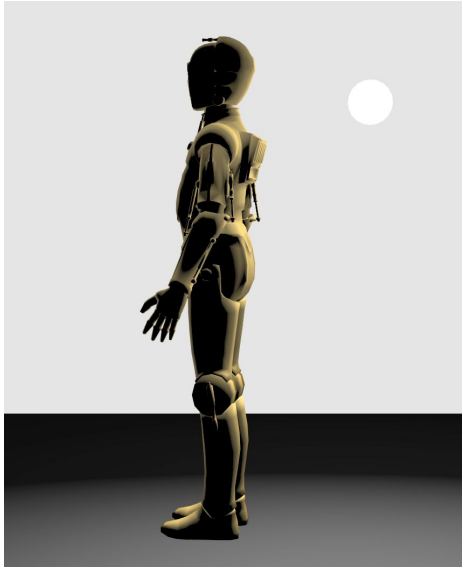
Task 4: Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```
{
  //TASK 3-6 create white light node at [0, 2, 2]
  let light = new LightNode();
  light.ambient = [0, 0, 0, 1];
  light.diffuse = [1, 1, 1, 1];
  light.specular = [1, 1, 1, 1];
  light.position = [0, 2, 2];
  light.append(createLightSphere());
  //TASK 4-1 animated light using rotateLight transformation node
  rotateLight = new TransformationSGNode(mat4.create(), [
      light
  ]);
  root.append(rotateLight);
}
```

```
123     //TASK 4-2 enable light rotation
124     rotateLight.matrix = glm.rotateY(timeInMilliseconds*0.05);
```





main.js

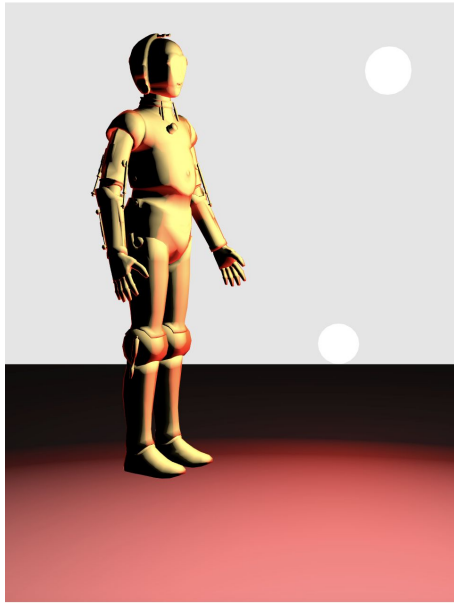
Extra Task 5: Multiple Light Sources

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Finally, what about multiple light sources?

Let's create a second one:

5-1 main.js: create 2nd red light node at [2, 0.2, 0]
5-2 main.js: rotate also this light node
→ Try the framework's Animation class



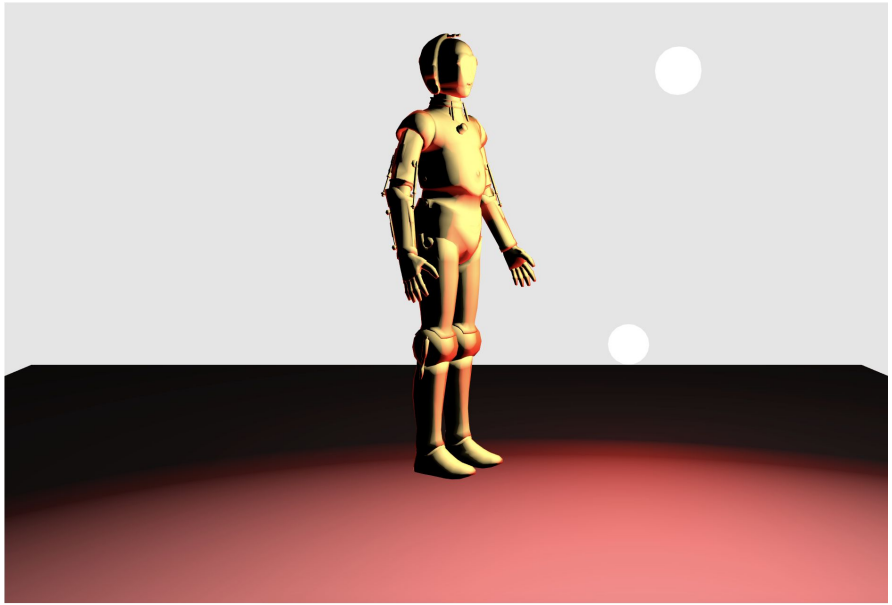
5-3, 5-4 vertex shader: consider 2nd light source
5-5, 5-6 fragment shader: consider 2nd light source

Extra Task 5: main.js Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```
54      {
55        //TASK 5-1 create red light node at [2, 0.2, 0]
56        let light2 = new LightNode();
57        light2.uniform = 'u_light2';
58        light2.diffuse = [1, 0, 0, 1];
59        light2.specular = [1, 0, 0, 1];
60        light2.position = [2, 0.2, 0];
61        light2.append(createLightSphere());
62        rotateLight2 = new TransformationSGNode(mat4.create(), [
63            light2
64        ]);
65        root.append(rotateLight2);
66      }
```

```
122
123     //TASK 4-2 enable light rotation
124     rotateLight.matrix = glm.rotateY(timeInMilliseconds*0.05);
125     //TASK 5-2 enable light rotation
126     rotateLight2.matrix = glm.rotateY(-timeInMilliseconds*0.1);
127
128     root.render(context);
```



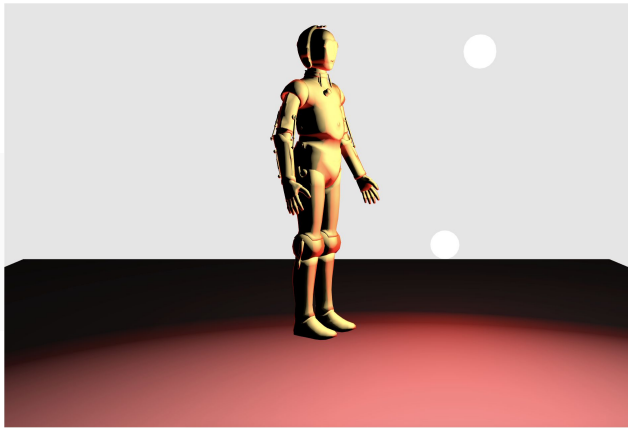
main.js

Extra Task 5: Shader Solution

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Vertex shader:

```
12    //TASK 3-3 light position as uniform
13    //vec3 lightPos = vec3(0, -2, 2);
14    uniform vec3 u_lightPos;
15    //TASK 5-3 second light source
16    uniform vec3 u_light2Pos;
17
```



Fragment shader:

```
34    //TASK 3-1 use uniform for light
35    //Light light = Light(vec4(0., 0., 0., 1.),
36    //                    vec4(1., 1., 1., 1.),
37    //                    vec4(1., 1., 1., 1.));
38    uniform Light u_light;
39    //TASK 5-5 use uniform for 2nd light
40    uniform Light u_light2;
41
```

```
29      v_eyeVec = -eyePosition.xyz;
30      //TASK 3-4 light position as uniform
31      v_lightVec = u_lightPos - eyePosition.xyz;
32      //TASK 5-4 second light source position
33      v_light2Vec = u_light2Pos - eyePosition.xyz;
34
35      gl_Position = u_projection * eyePosition;
36    }
```

```
70    void main() {
71      //TASK 2-3 use material uniform
72      //TASK 3-2 use light uniform
73      //TASK 5-6 use second light source
74      gl_FragColor =
75        calculateSimplePointLight(u_light, u_material, v_lightVec, v_normalVec, v_eyeVec)
76        + calculateSimplePointLight(u_light2, u_material, v_light2Vec, v_normalVec, v_eyeVec);
77
78    }
```

phong.vs.glsl

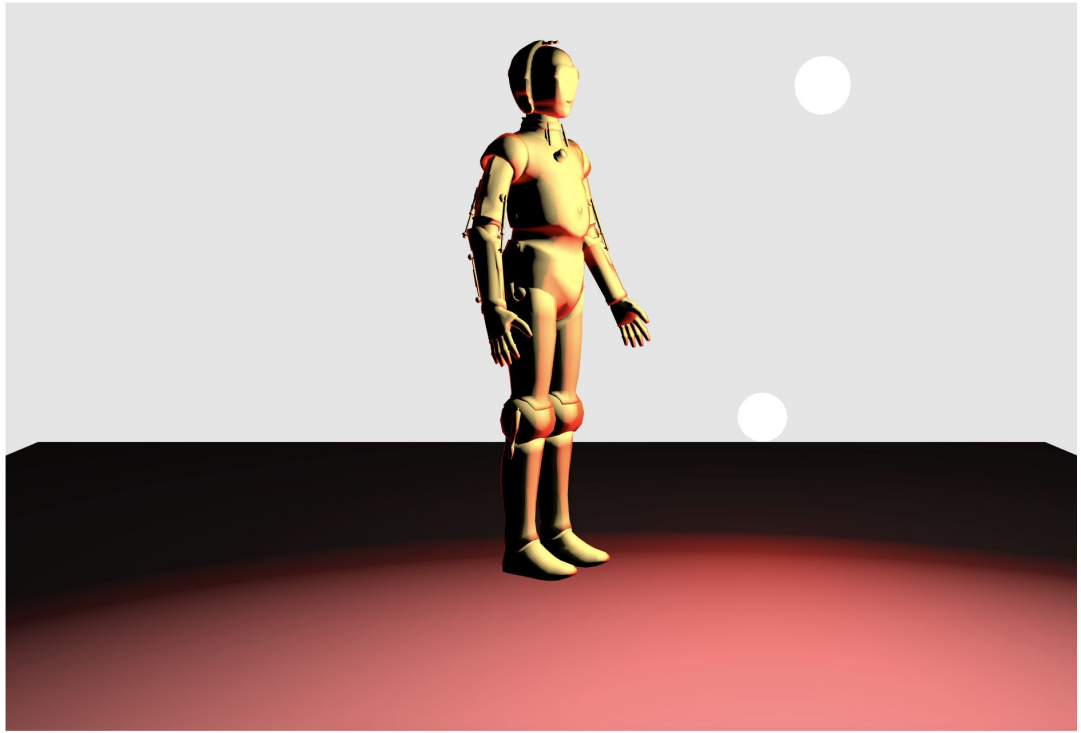
phong.fs.glsl

Recap

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Illumination

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

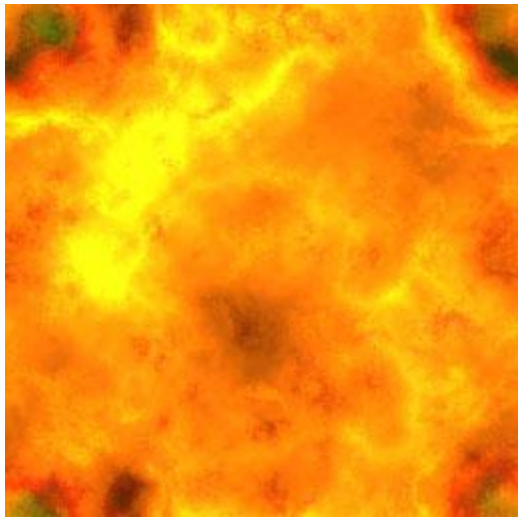


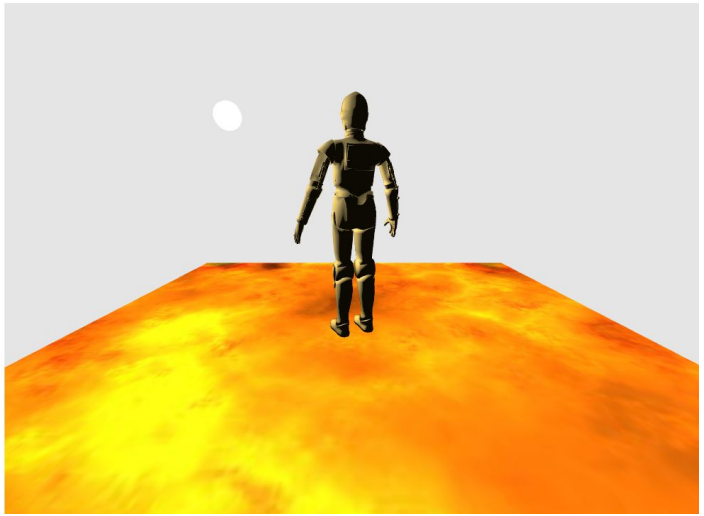
Next Time

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Texturing

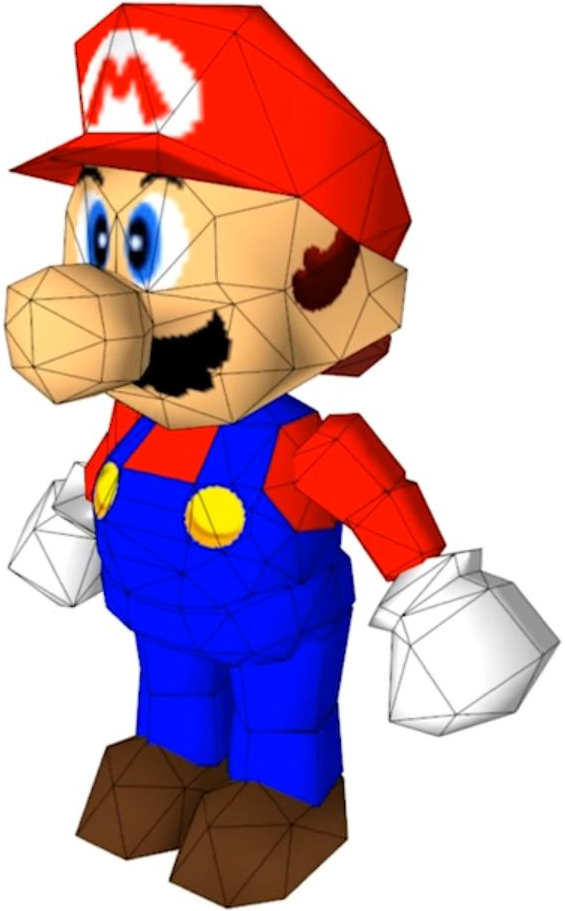
How to map an image on geometry?



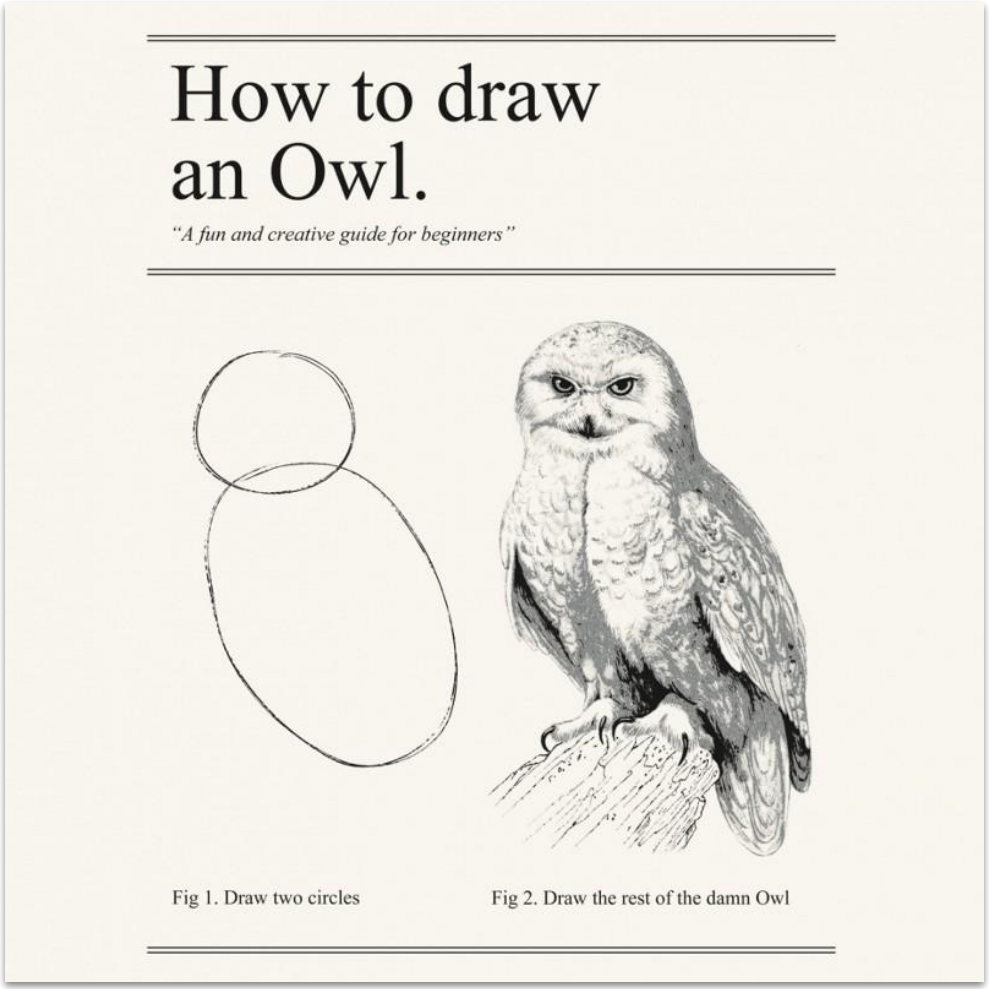


SUPER MARIO 64 - 1996
NINTENDO 64

TRIS - 752
FACES - 752
VERTS - 406



https://www.youtube.com/watch?v=A2gXyEyy_2U



JOHANNES KEPLER UNIVERSITY LINZ

INSTITUTE OF COMPUTER GRAPHICS

Thanks!
Have fun with your CG-Projects.

Questions / Feedback:
cg-lab@jku.at