JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

# Computer Graphics

## Lab 4: Illumination and Shading

# Schedule

| | | | |
|---|---|---|---|
| Lab 1 | Introduction to WebGL | 10.3.2017 / 14.3/2017 | - |
| Lab 2 | Transformations and Projections | 21.3.2017 / 24.3.2017 | Lecture: 14.3.2017 |
| Lab 3 | Scene Graphs | 28.3.2017 / 31.3.2017 | Lecture: 21.3.2017 |
| Lab 4 | Illumination and Shading | 4.4.2017 / 7.4.2017 | Lecture: 28.3.2017 |
| Lab 5 | Texturing | 25.4.2017 / 28.4.2017 | Lecture: 4.4.2017 |
| Lab 6 | Advanced Texture Mapping | 2.5.2017 / 5.5.2017 | Lecture: 26.4.2017 |
| Lab 7a | CUDA | 9.5.2017 / 12.5.2017 | |
| Lab 7b | VTK | 12.5.2017 | |

# Slides and lab material

www.cg.jku.at/teaching/computergraphics/lab

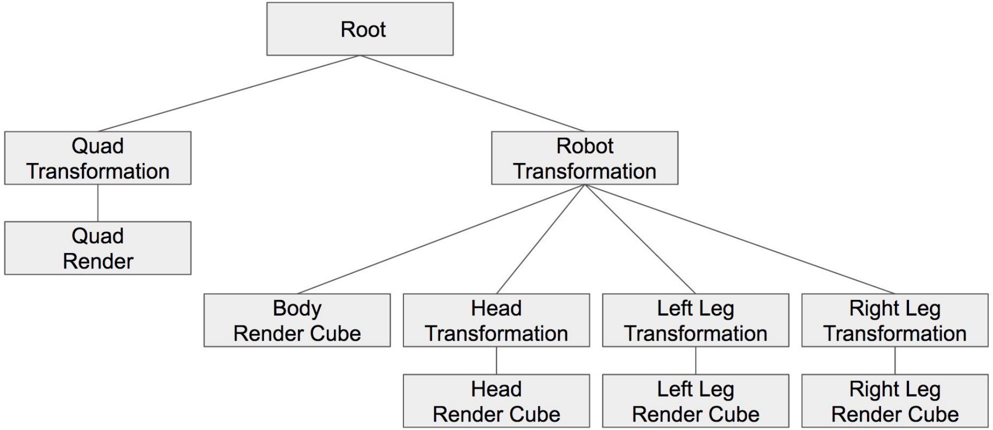
# Recap

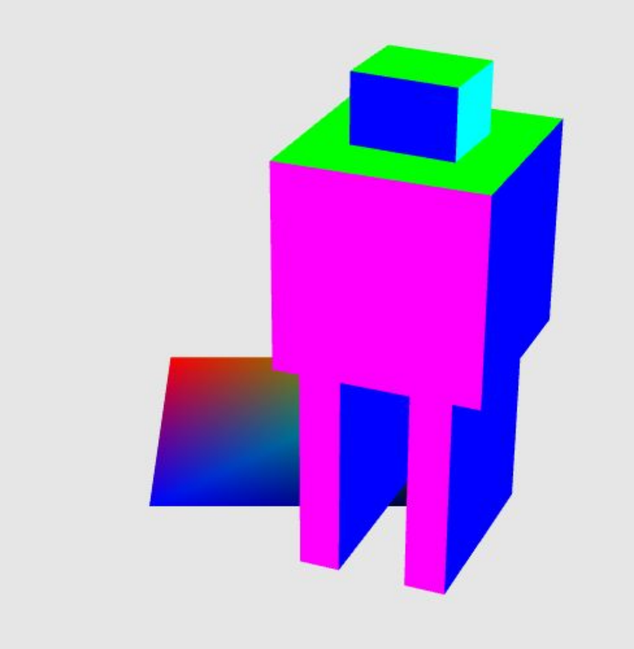
Depth Handling

Blending

Scene Graphs

- Abstraction into Nodes
- Scene graph traversal
- Implement robot using a scene graph

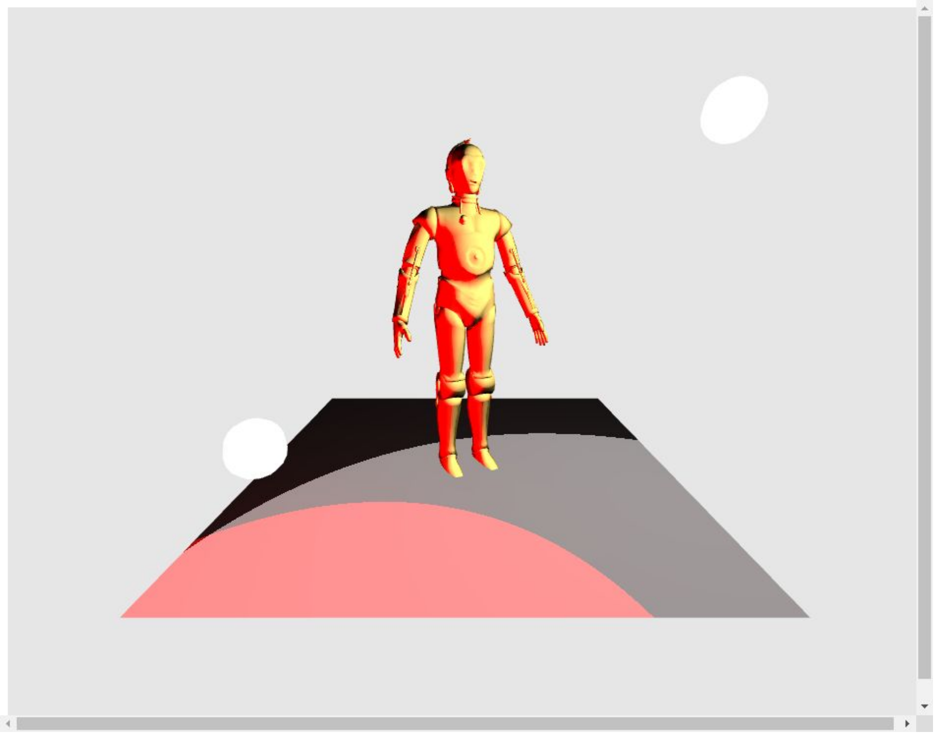




# Agenda for Today

Illumination

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



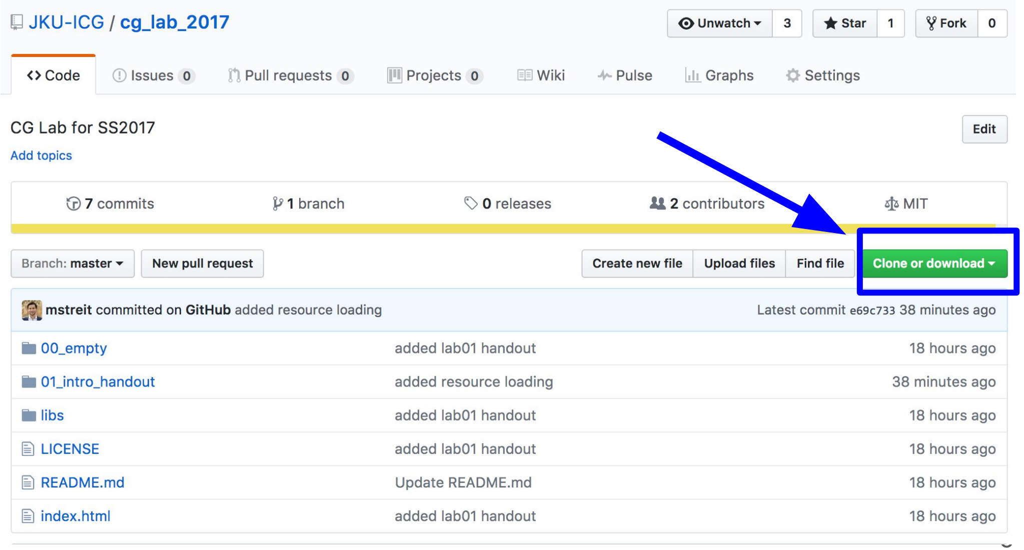
# Dev Environment: Lab Package

Hosted on GitHub: https://github.com/jku-icg/cg_lab_2017

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 Atom editor
4. Use “Open Folder” in Atom
5. Start server on any port (Packages -> Live Server)

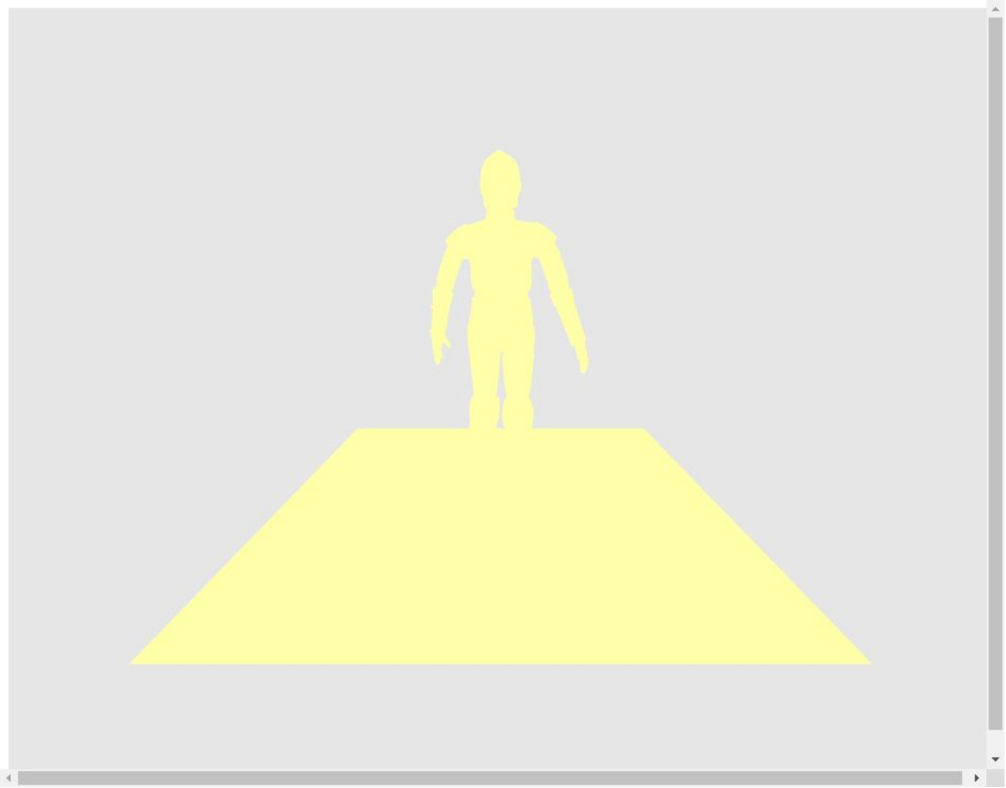


CG Lab for SS2017

mstreit committed on GitHub added resource loading — Latest commit e69c733 38 minutes ago

| | | |
|---|---|---|
| 00_empty | added lab01 handout | 18 hours ago |
| 01_intro_handout | added resource loading | 38 minutes ago |
| libs | added lab01 handout | 18 hours ago |
| LICENSE | added lab01 handout | 18 hours ago |
| README.md | Update README.md | 18 hours ago |
| index.html | added lab01 handout | 18 hours ago |

# Why do we need it?



Not really 3D, right?

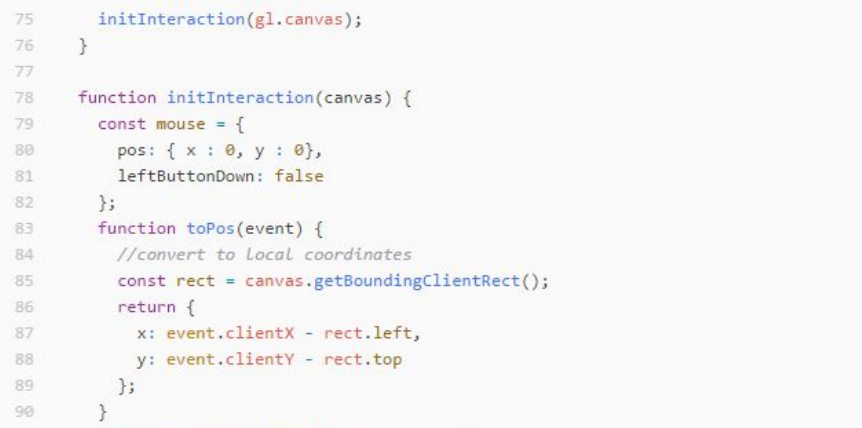
Looks flat and boring

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

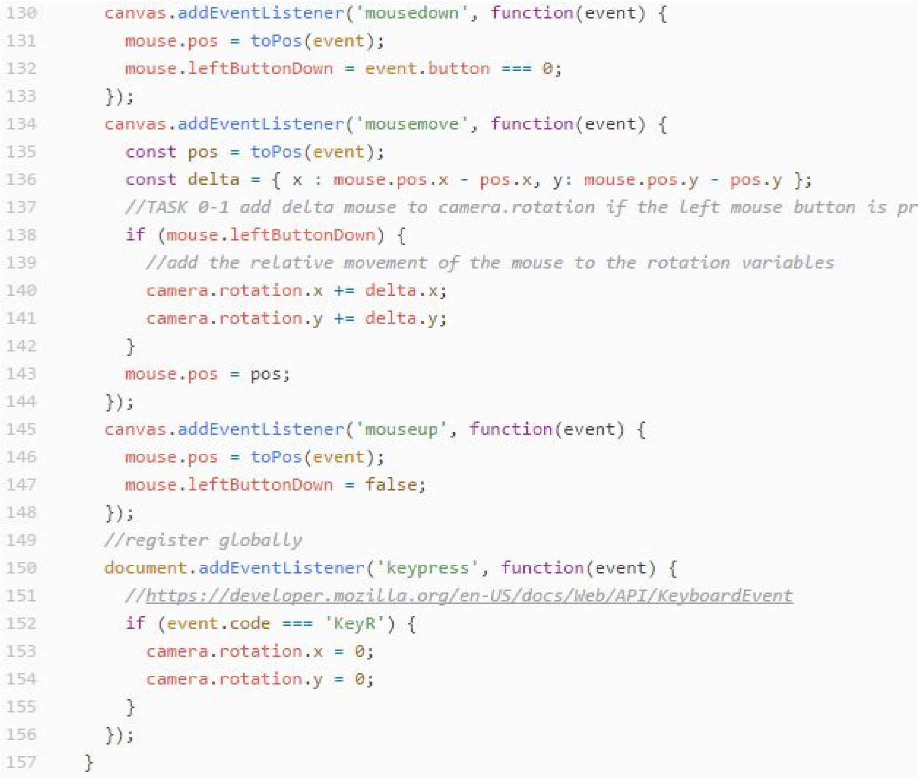
# WebGL Interaction

## HTML Event listener

mousedown, mousemove, mouseup
keypress, keydown, keyup
....



```
  initInteraction(gl.canvas);
}

function initInteraction(canvas) {
  const mouse = {
    pos: { x : 0, y : 0},
    leftButtonDown: false
  };
  function toPos(event) {
    //convert to local coordinates
    const rect = canvas.getBoundingClientRect();
    return {
      x: event.clientX - rect.left,
      y: event.clientY - rect.top
    };
  }
```



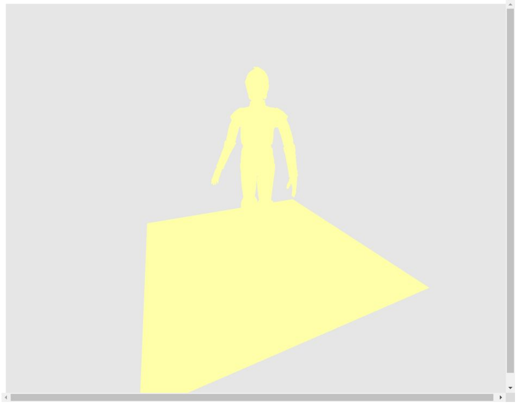
```
  canvas.addEventListener('mousedown', function(event) {
    mouse.pos = toPos(event);
    mouse.leftButtonDown = event.button === 0;
  });
  canvas.addEventListener('mousemove', function(event) {
    const pos = toPos(event);
    const delta = { x : mouse.pos.x - pos.x, y: mouse.pos.y - pos.y };
    //TASK 0-1 add delta mouse to camera.rotation if the left mouse button is pr
    if (mouse.leftButtonDown) {
      //add the relative movement of the mouse to the rotation variables
      camera.rotation.x += delta.x;
      camera.rotation.y += delta.y;
    }
    mouse.pos = pos;
  });
  canvas.addEventListener('mouseup', function(event) {
    mouse.pos = toPos(event);
    mouse.leftButtonDown = false;
  });
  //register globally
  document.addEventListener('keypress', function(event) {
    //https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent
    if (event.code === 'KeyR') {
      camera.rotation.x = 0;
      camera.rotation.y = 0;
    }
  });
}
```

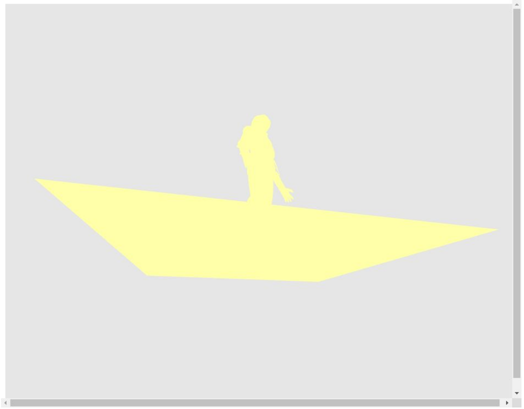
# Task 0: Rotate Scene with Mouse

e.g., rotate scene with mouse stored in `camera.rotation.x` and `camera.rotation.y`

0-1: within the `mousemove` event listener, change according to the delta mouse position if left mouse button is pressed

0-2: set a rotated scene matrix in `render()` accordingly

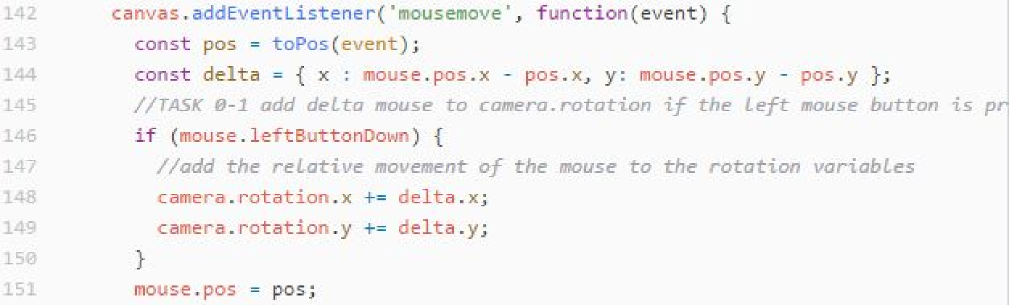




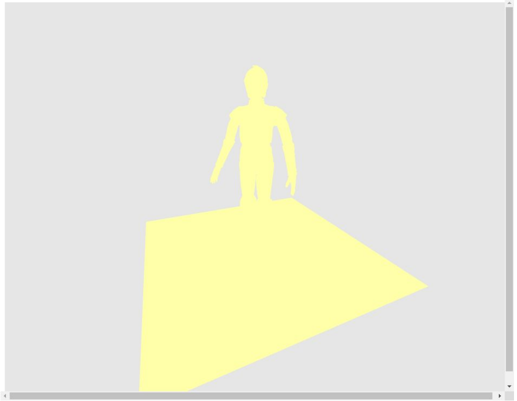
# Task 0: Solution

## e.g., rotate scene with mouse

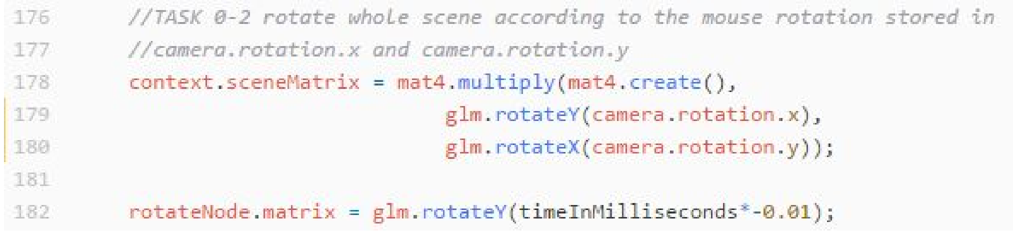
`initInteraction(canvas)`



```
canvas.addEventListener('mousemove', function(event) {
  const pos = toPos(event);
  const delta = { x : mouse.pos.x - pos.x, y: mouse.pos.y - pos.y };
  //TASK 0-1 add delta mouse to camera.rotation if the left mouse button is pr
  if (mouse.leftButtonDown) {
    //add the relative movement of the mouse to the rotation variables
    camera.rotation.x += delta.x;
    camera.rotation.y += delta.y;
  }
  mouse.pos = pos;
```



`render()`

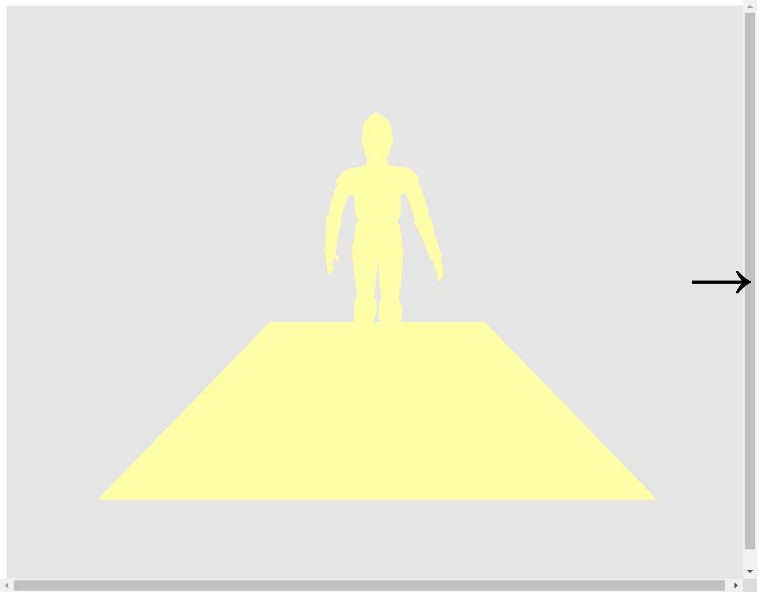


```
//TASK 0-2 rotate whole scene according to the mouse rotation stored in
//camera.rotation.x and camera.rotation.y
context.sceneMatrix = mat4.multiply(mat4.create(),
                          glm.rotateY(camera.rotation.x),
                          glm.rotateX(camera.rotation.y));

rotateNode.matrix = glm.rotateY(timeInMilliseconds*-0.01);
```

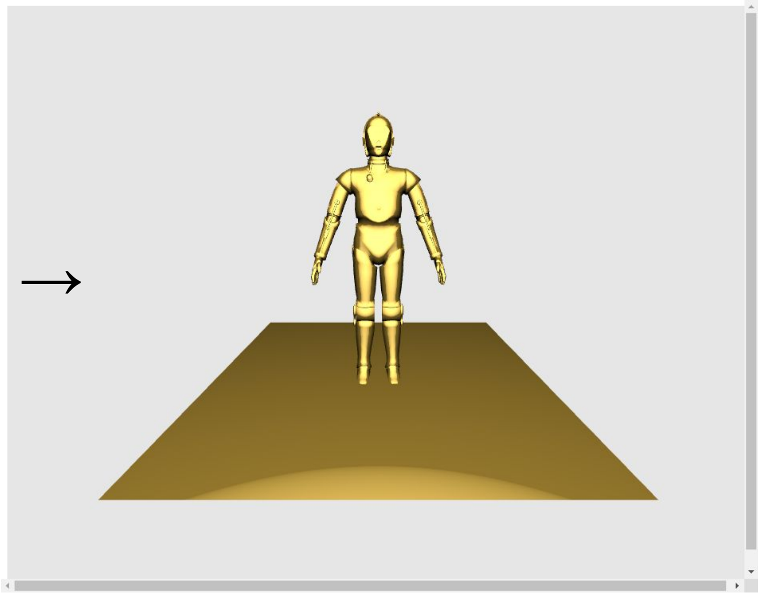
It is 3D :)

But we cannot see it in the shading

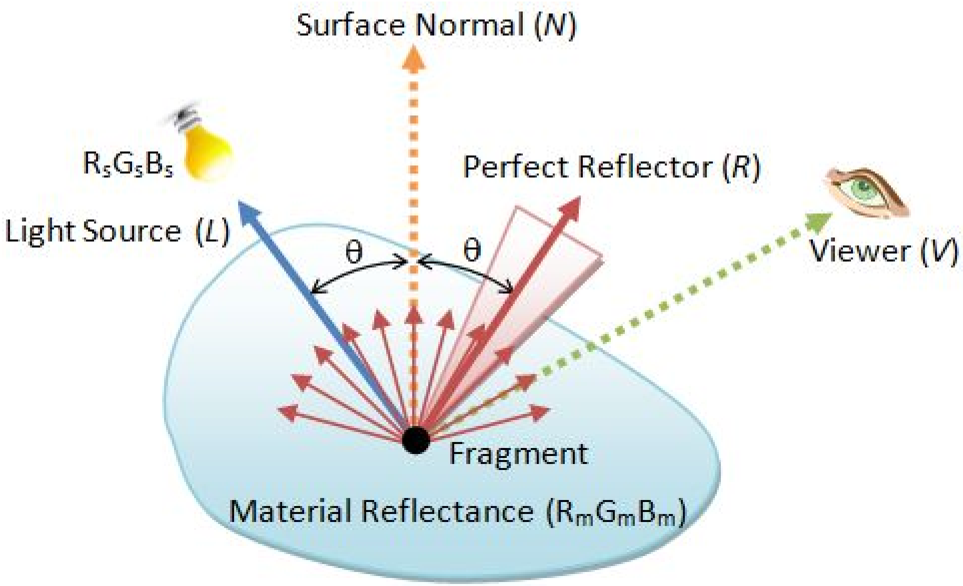
# What is missing?



→ Illumination →



# Phong-Shading



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

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

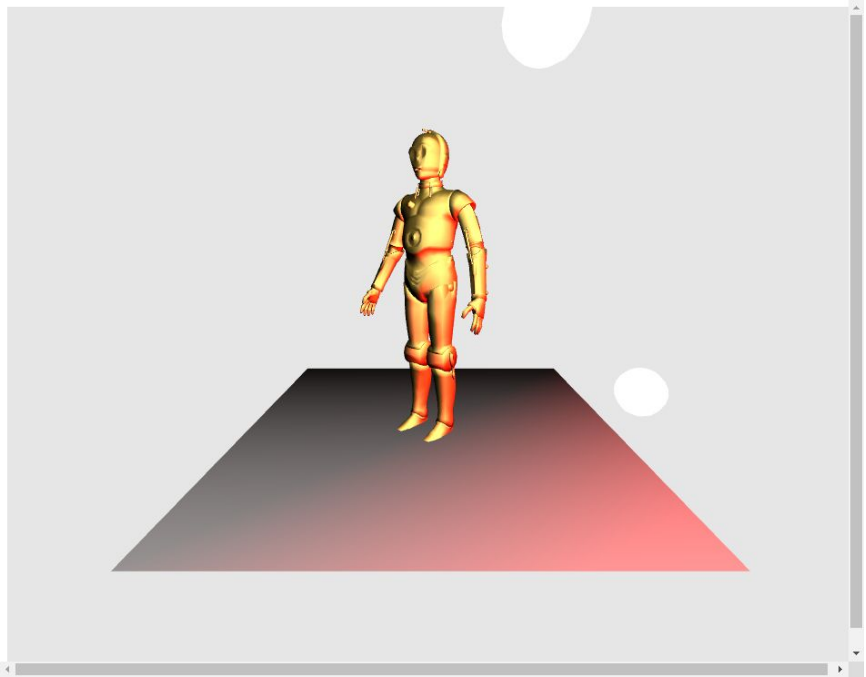
$$c_{spec} = max(R \cdot V, 0)^{m\_sh} * l_{spec} * m_{spec}$$

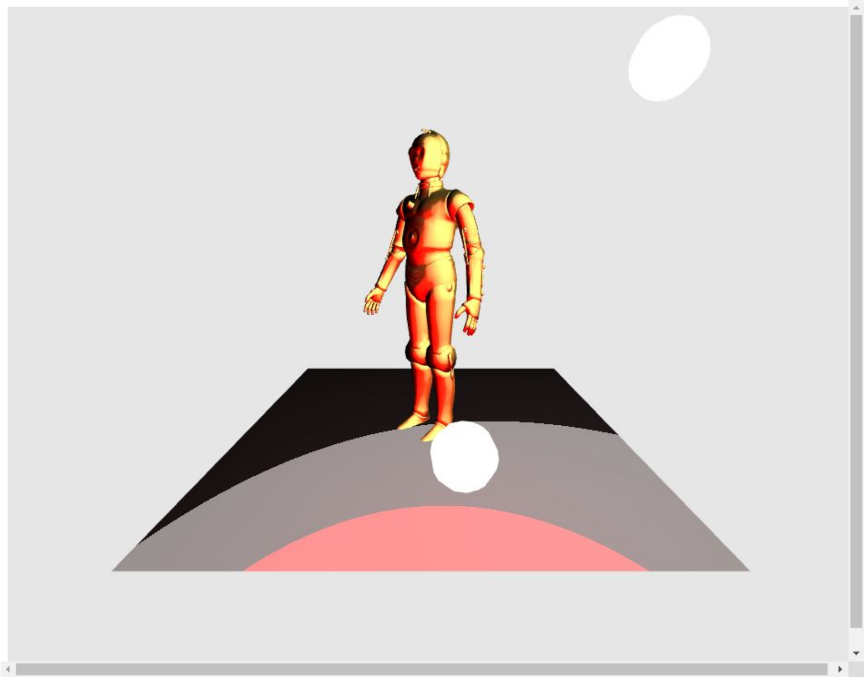
$$c_{em} = m_{em}$$

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

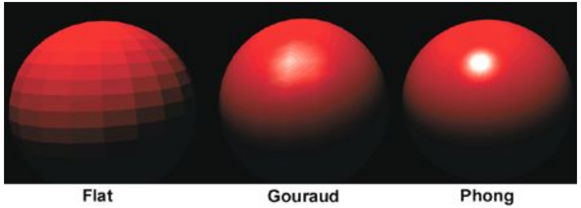
http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

# Gouraud vs. Phong Shading



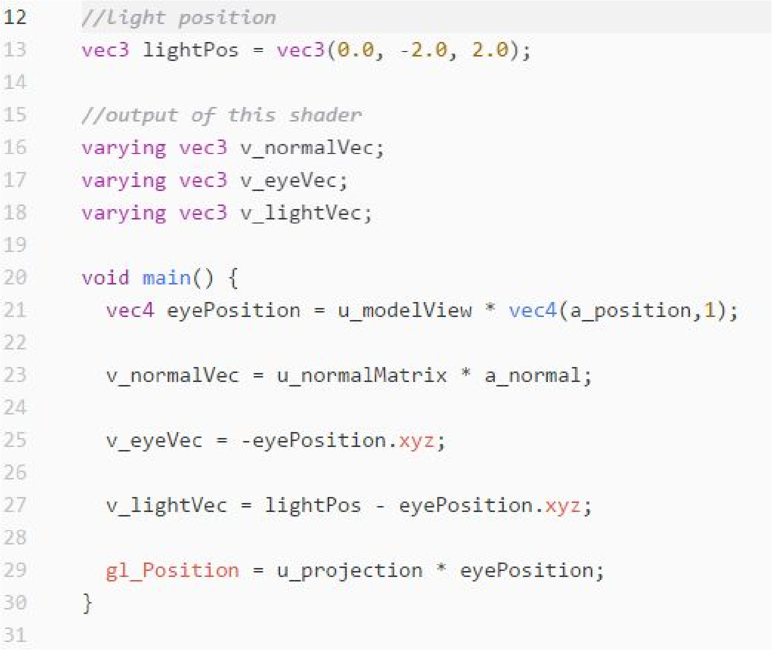


Can you spot & explain the difference?



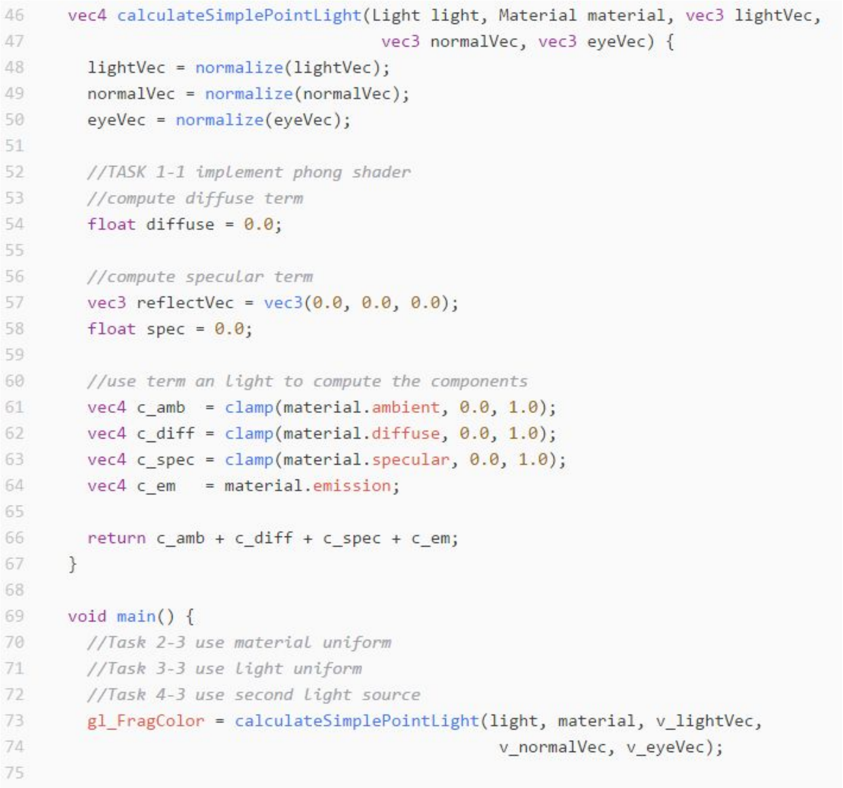
# Phong Shader

## Vertex Shader:



```
//light position
vec3 lightPos = vec3(0.0, -2.0, 2.0);

//output of this shader
varying vec3 v_normalVec;
varying vec3 v_eyeVec;
varying vec3 v_lightVec;

void main() {
  vec4 eyePosition = u_modelView * vec4(a_position,1);

  v_normalVec = u_normalMatrix * a_normal;

  v_eyeVec = -eyePosition.xyz;

  v_lightVec = lightPos - eyePosition.xyz;

  gl_Position = u_projection * eyePosition;
}
```

## Fragment Shader:



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

  //TASK 1-1 implement phong shader
  //compute diffuse term
  float diffuse = 0.0;

  //compute specular term
  vec3 reflectVec = vec3(0.0, 0.0, 0.0);
  float spec = 0.0;

  //use term an light to compute the components
  vec4 c_amb  = clamp(material.ambient, 0.0, 1.0);
  vec4 c_diff = clamp(material.diffuse, 0.0, 1.0);
  vec4 c_spec = clamp(material.specular, 0.0, 1.0);
  vec4 c_em   = material.emission;

  return c_amb + c_diff + c_spec + c_em;
}

void main() {
  //Task 2-3 use material uniform
  //Task 3-3 use light uniform
  //Task 4-3 use second light source
  gl_FragColor = calculateSimplePointLight(light, material, v_lightVec,
                                           v_normalVec, v_eyeVec);
```

# Task 1: Implement Phong Shader

## 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.*

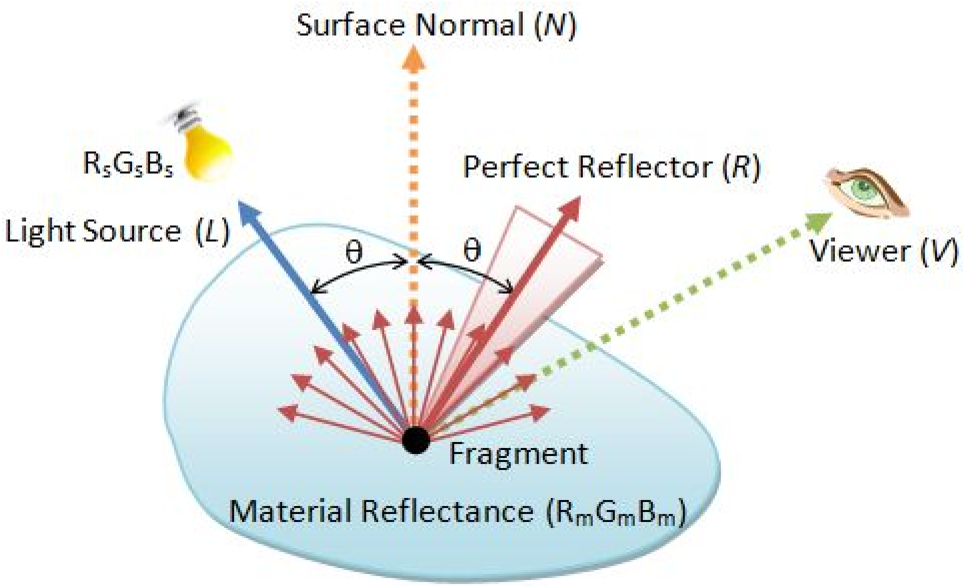
`clamp(x,minVal,maxVal)` - returns min (max (*x, minVal), maxVal).*

`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.

# Task 1: Implement Phong Shader



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

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

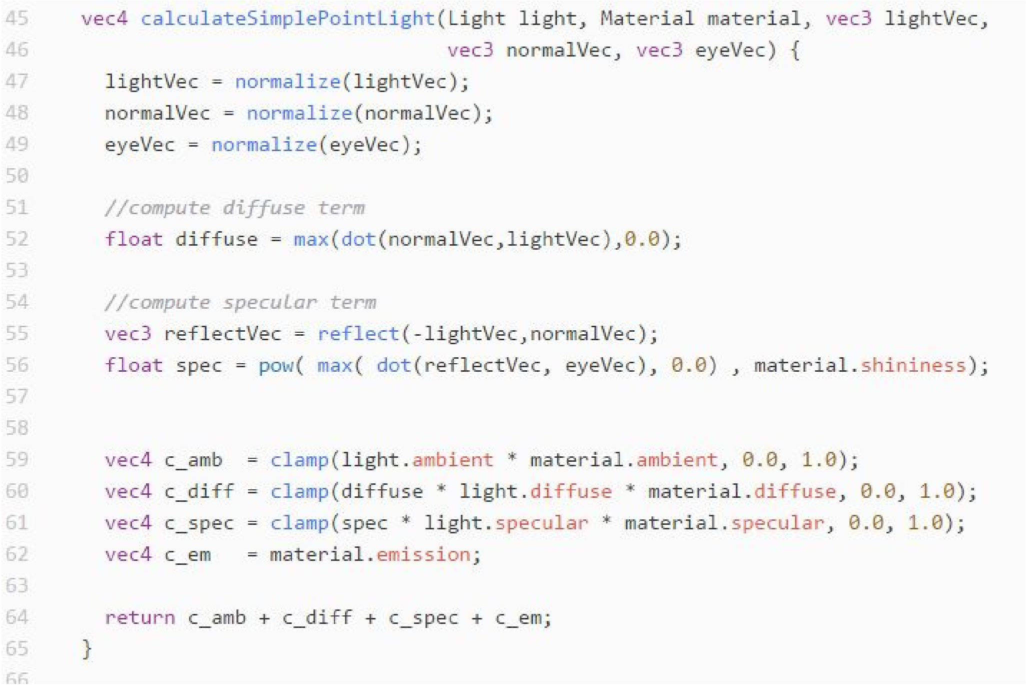
$$c_{spec} = max(R \cdot V, 0)^{m\_sh} * l_{spec} * m_{spec}$$

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

$$\mathbf{c}_{\mathbf{final}} = c_{amb} + c_{diff} + c_{spec} + c_{em}$$

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

# Task 1: Solution



```
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;
}
```

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

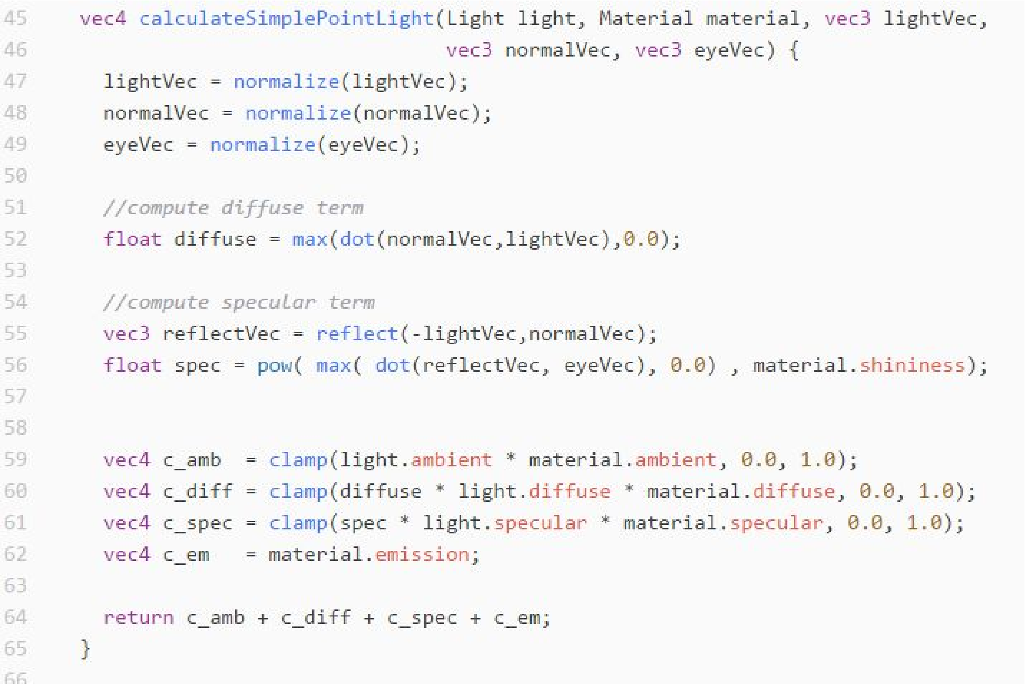
$$c_{diff} = max(L \cdot N, 0) * l_{diff} * m_{diff}$$

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

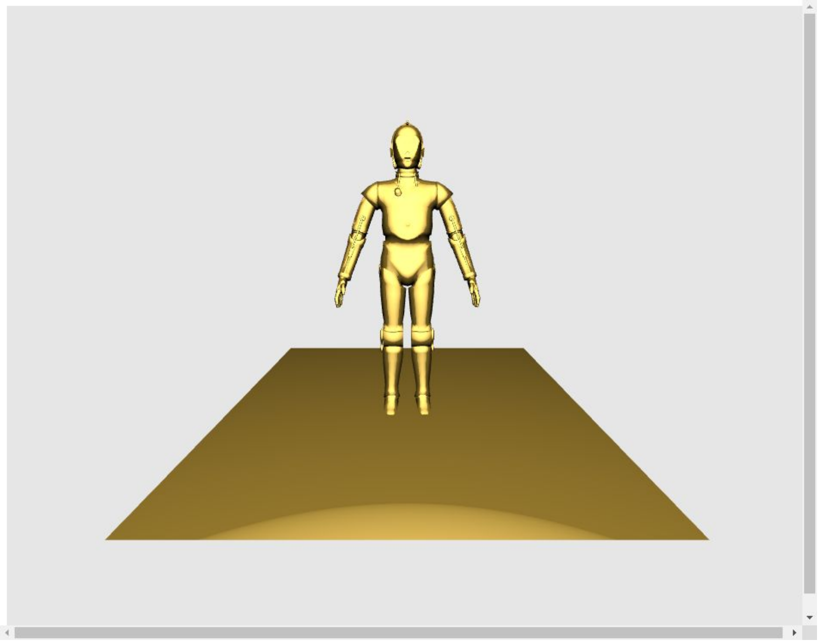
$$c_{em} = m_{em}$$

$$\mathbf{c}_{\mathbf{final}} = c_{amb} + c_{diff} + c_{spec} + c_{em}$$

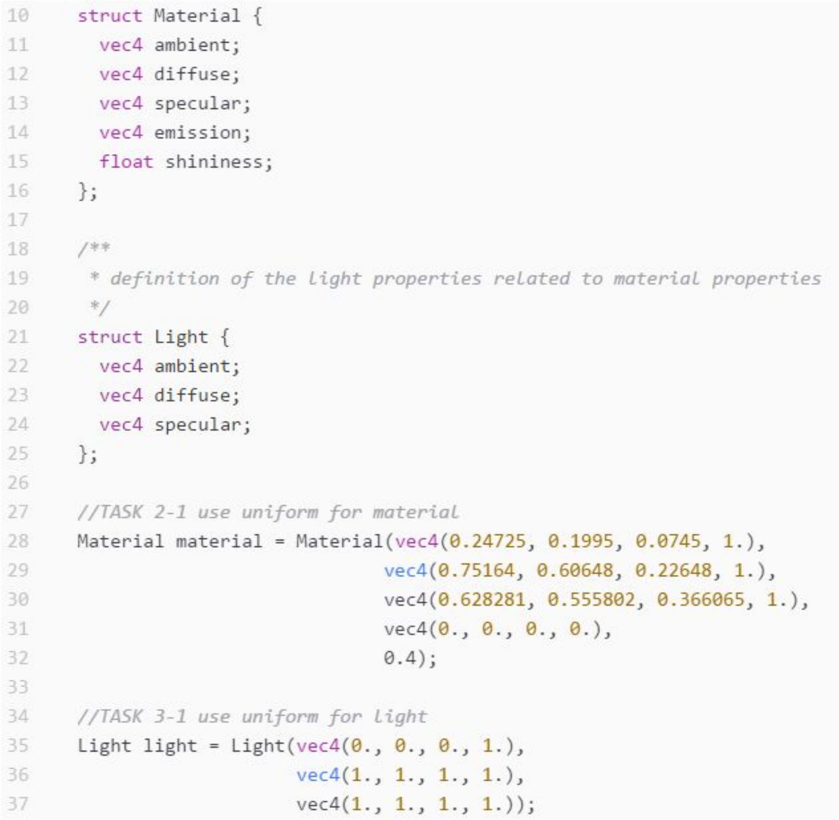
# Task 1: Solution



```
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;
}
```



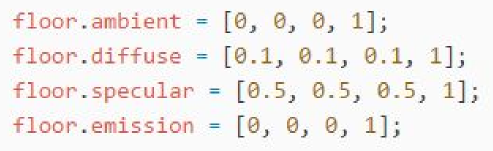
# Task 2: Extract to MaterialNode



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

/**
 * definition of the light properties related to material properties
 */
struct Light {
  vec4 ambient;
  vec4 diffuse;
  vec4 specular;
};

//TASK 2-1 use uniform for material
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);

//TASK 3-1 use uniform for light
Light light = Light(vec4(0., 0., 0., 1.),
                    vec4(1., 1., 1., 1.),
                    vec4(1., 1., 1., 1.));
```

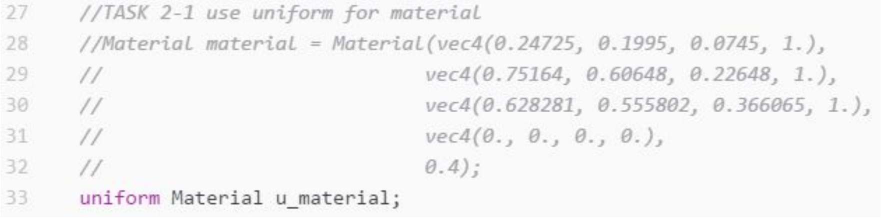
Hard coded material and light properties...

2-1, 2-2 fragment shader:
define uniform: `u_material` and use it
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 one
2-5 main.js: wrap floor with material, too

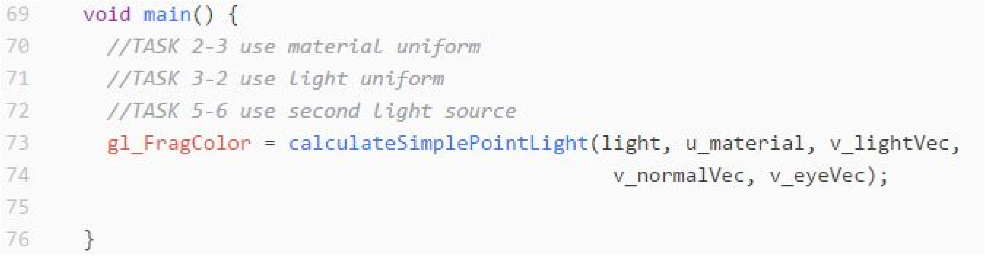


```
floor.ambient = [0, 0, 0, 1];
floor.diffuse = [0.1, 0.1, 0.1, 1];
floor.specular = [0.5, 0.5, 0.5, 1];
floor.emission = [0, 0, 0, 1];
```

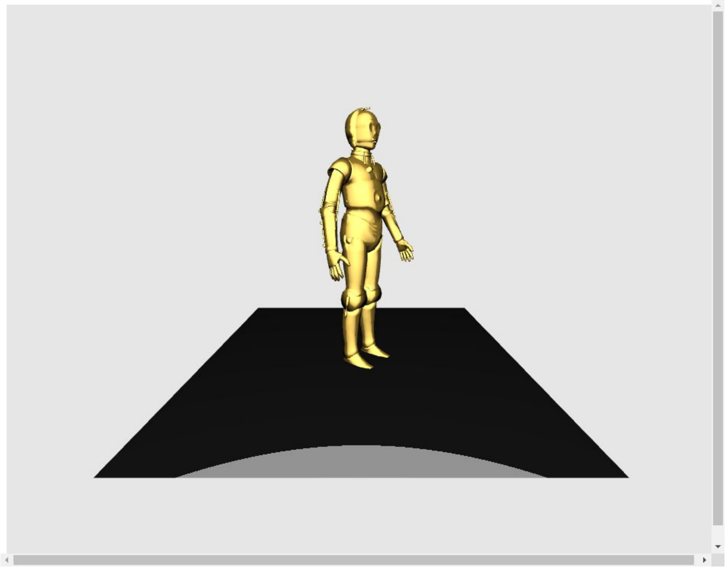
# Task 2: Fragment Shader Solution



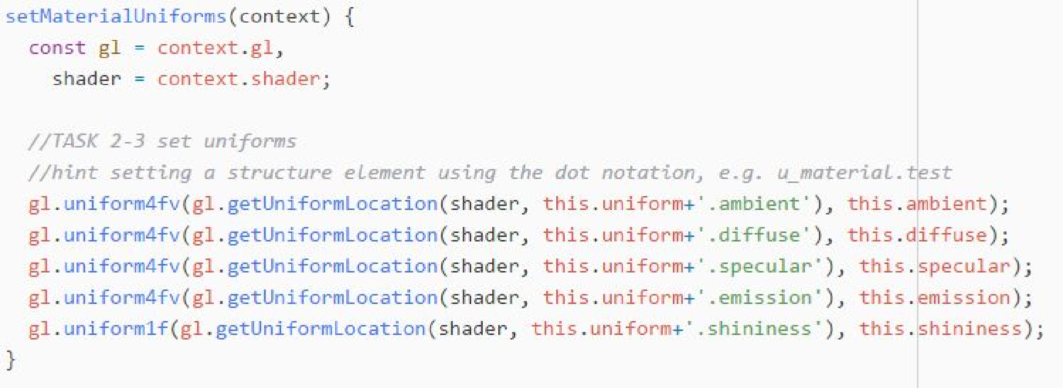
```glsl
//TASK 2-1 use uniform for material
//Material material = Material(vec4(0.24725, 0.1995, 0.0745, 1.),
//                             vec4(0.75164, 0.60648, 0.22648, 1.),
//                             vec4(0.628281, 0.555802, 0.366065, 1.),
//                             vec4(0., 0., 0., 0.),
//                             0.4);
uniform Material u_material;
```



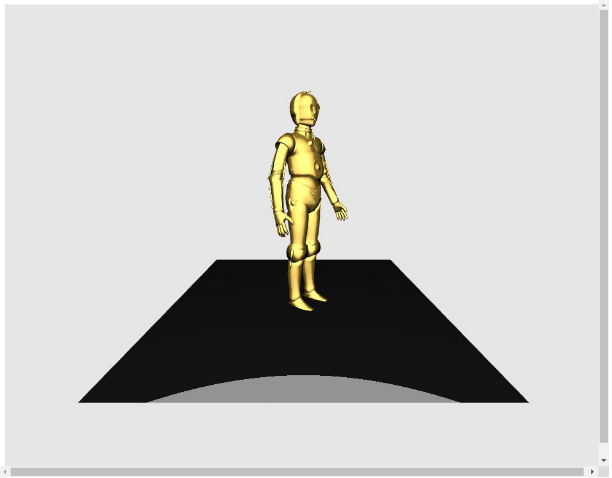
```glsl
void main() {
  //TASK 2-3 use material uniform
  //TASK 3-2 use light uniform
  //TASK 5-6 use second light source
  gl_FragColor = calculateSimplePointLight(light, u_material, v_lightVec,
                                           v_normalVec, v_eyeVec);

}
```

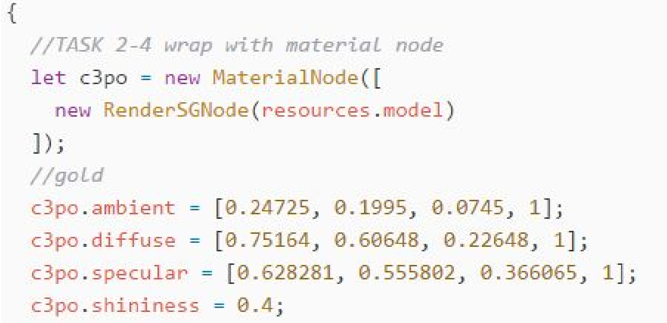


# Task 2: main.js Solution

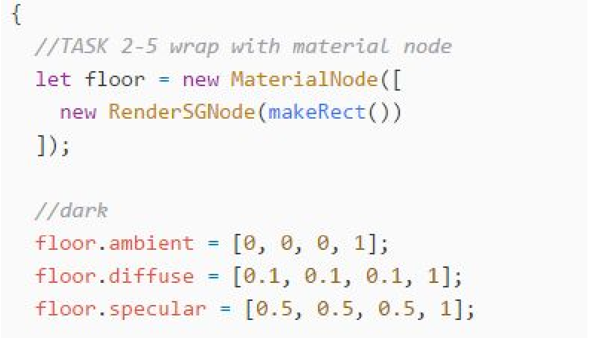


```
setMaterialUniforms(context) {
  const gl = context.gl,
    shader = context.shader;

  //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];
```

# Task 3: Extract Light Node

Same game 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

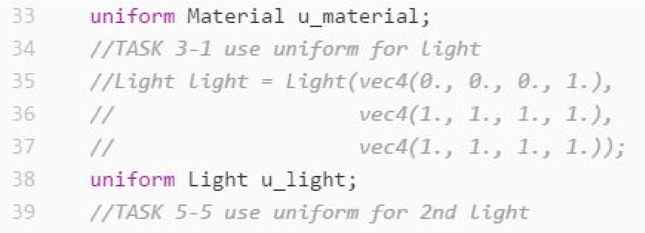
3-5 main.js: finish `LightNode` class

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

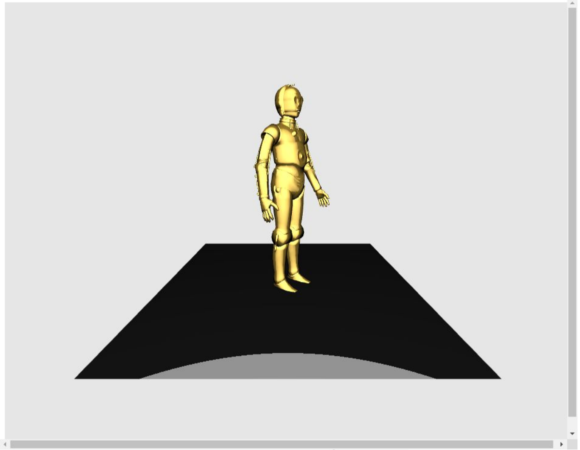
  + append as a child the return value of `createLightSphere()`

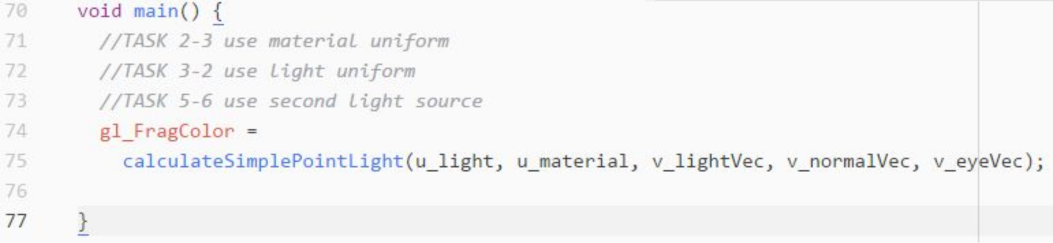
# Task 3: Shader Solution

## Fragment shader:



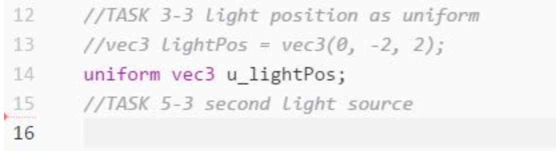
```
uniform Material u_material;
//TASK 3-1 use uniform for light
//Light light = Light(vec4(0., 0., 0., 1.),
//                    vec4(1., 1., 1., 1.),
//                    vec4(1., 1., 1., 1.));
uniform Light u_light;
//TASK 5-5 use uniform for 2nd light
```



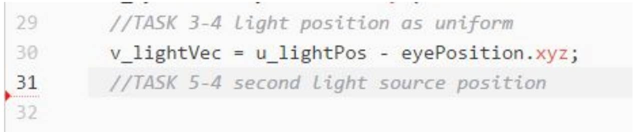


```
void main() {
  //TASK 2-3 use material uniform
  //TASK 3-2 use light uniform
  //TASK 5-6 use second light source
  gl_FragColor =
    calculateSimplePointLight(u_light, u_material, v_lightVec, v_normalVec, v_eyeVec);

}
```

## Vertex shader:

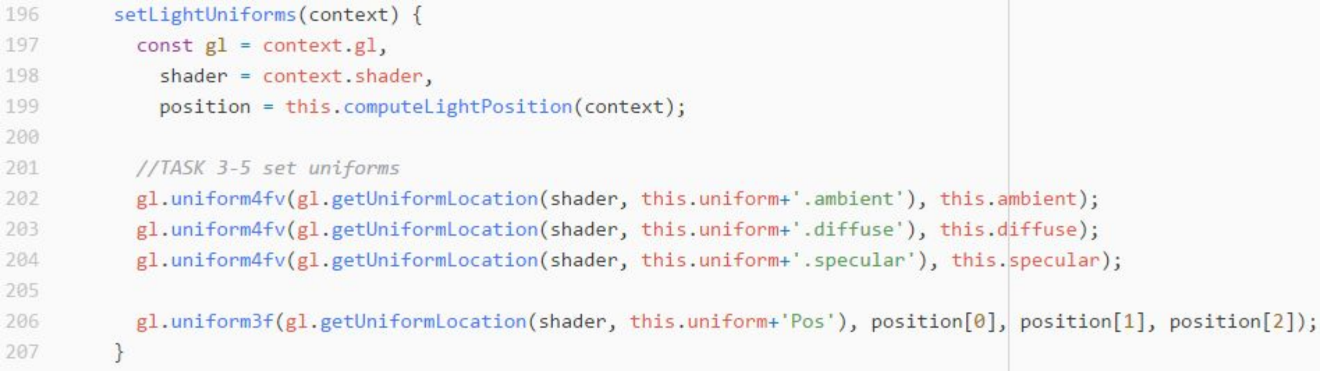


```
//TASK 3-3 light position as uniform
//vec3 lightPos = vec3(0, -2, 2);
uniform vec3 u_lightPos;
//TASK 5-3 second light source

```

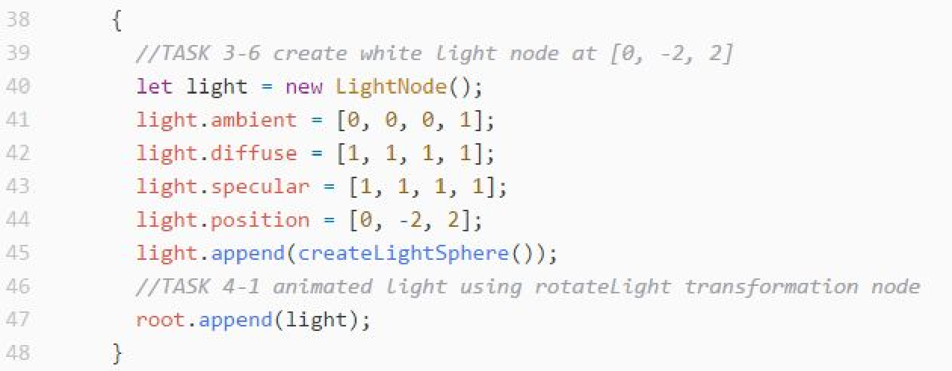


```
  //TASK 3-4 light position as uniform
  v_lightVec = u_lightPos - eyePosition.xyz;
  //TASK 5-4 second light source position

```

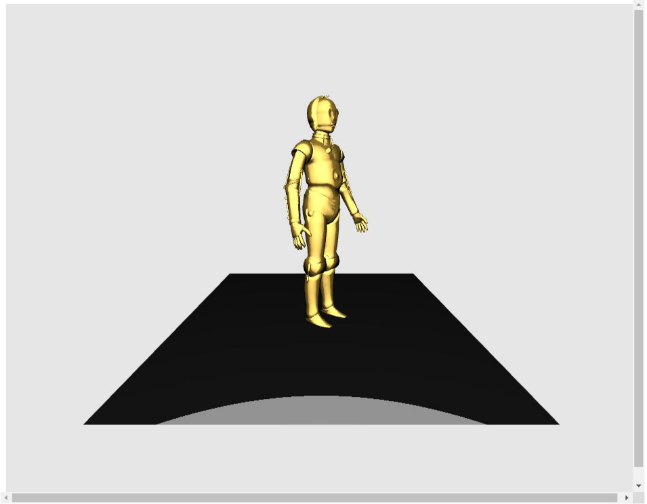
# Task 3: main.js Solution



```
setLightUniforms(context) {
  const gl = context.gl,
    shader = context.shader,
    position = this.computeLightPosition(context);

  //TASK 3-5 set uniforms
  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.uniform3f(gl.getUniformLocation(shader, this.uniform+'Pos'), position[0], position[1], position[2]);
}
```



```
{
  //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
  root.append(light);
}
```



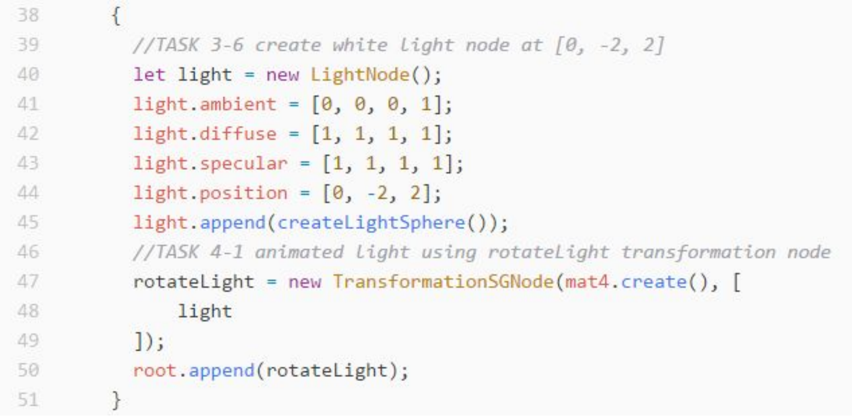
# Task 4: Animate Light Source

Static lights are boring … let’s animate the light source

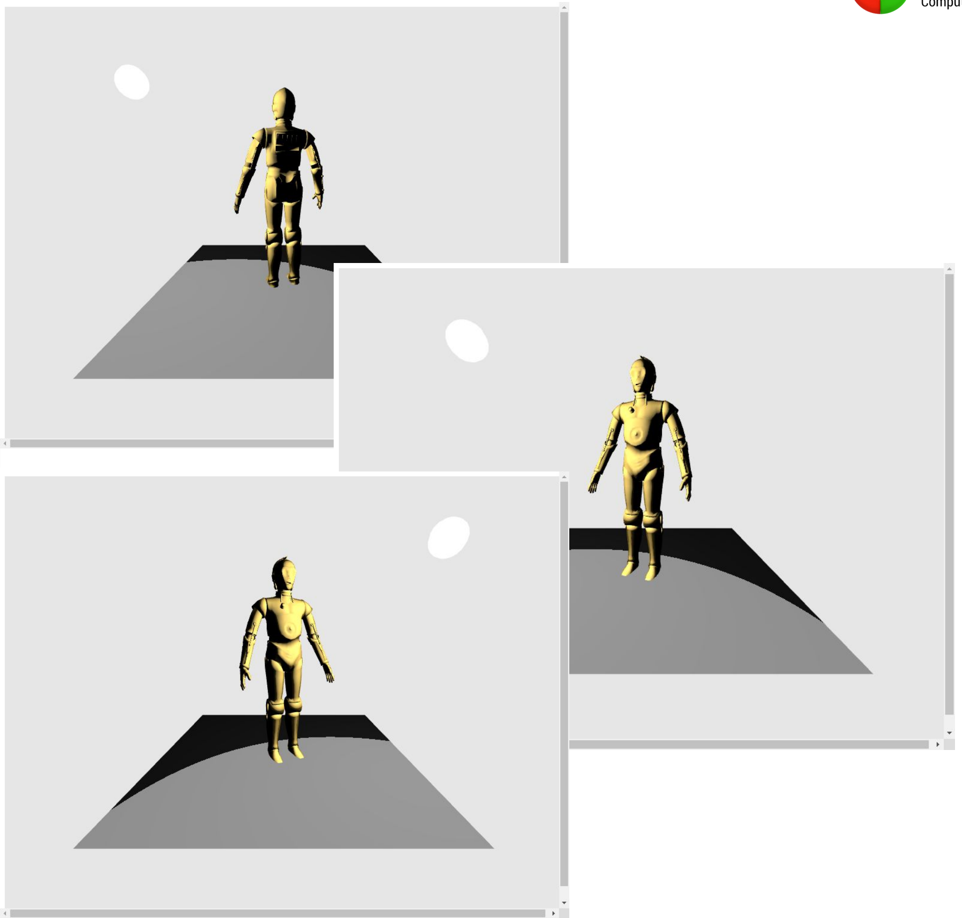
4-1 main.js: wrap the light node with a transformation
node and store it in `rotateLight`

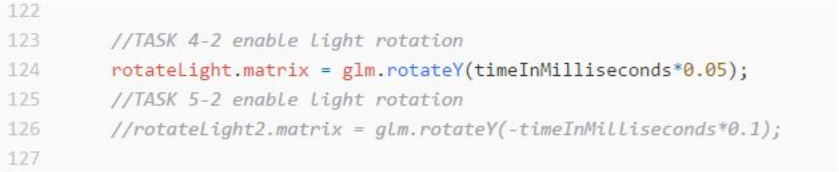
4-2 main.js: enable animation of `rotateLight` in the render method

# Task 4: Solution



```
{
  //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);
}
```





```

//TASK 4-2 enable light rotation
rotateLight.matrix = glm.rotateY(timeInMilliseconds*0.05);
//TASK 5-2 enable light rotation
//rotateLight2.matrix = glm.rotateY(-timeInMilliseconds*0.1);

```

# Extra Task 5: Multiple Light Sources

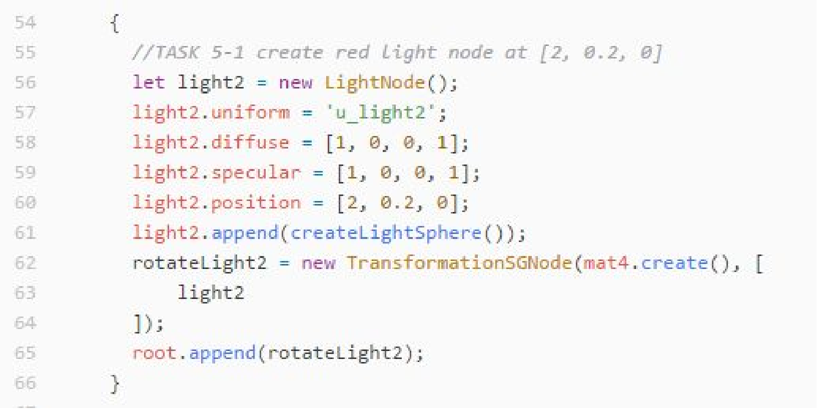
Finally, what about multiple light sources

Let’s create a second one:

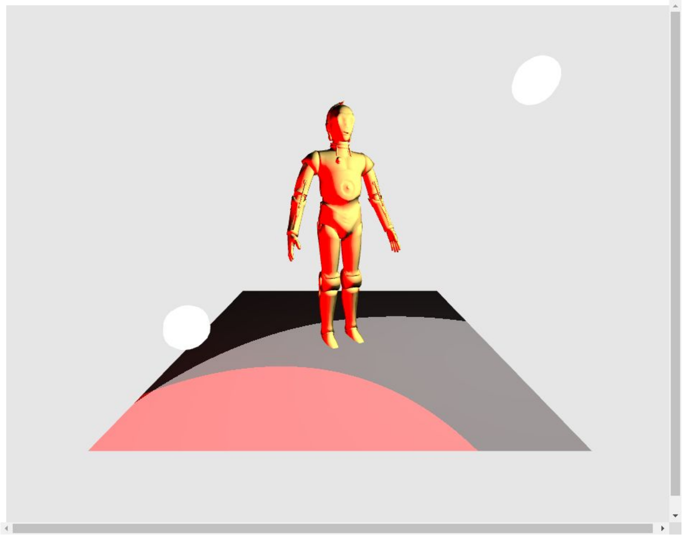
5-1 main.js: create 2nd red light node at [2, 0.2, 0]
5-2 main.js: rotate also this light node

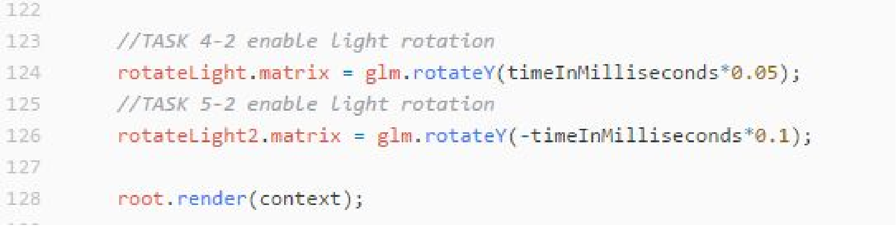
5-3, 5-4 vertex shader: consider 2nd light source
5-5, 5-6 fragment shader: consider 2nd light source

# Extra Task 5: main.js Solution



```
{
  //TASK 5-1 create red light node at [2, 0.2, 0]
  let light2 = new LightNode();
  light2.uniform = 'u_light2';
  light2.diffuse = [1, 0, 0, 1];
  light2.specular = [1, 0, 0, 1];
  light2.position = [2, 0.2, 0];
  light2.append(createLightSphere());
  rotateLight2 = new TransformationSGNode(mat4.create(), [
      light2
  ]);
  root.append(rotateLight2);
}
```

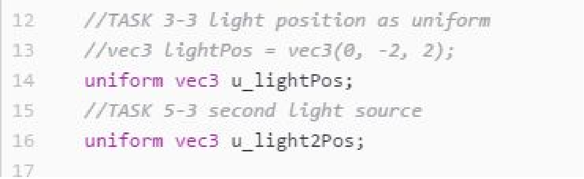




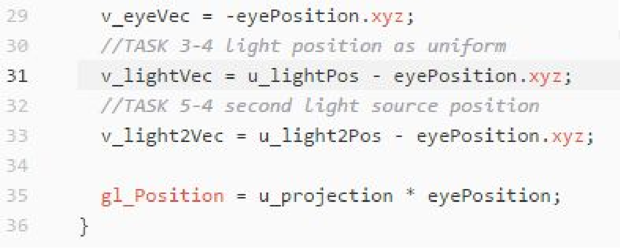
```

//TASK 4-2 enable light rotation
rotateLight.matrix = glm.rotateY(timeInMilliseconds*0.05);
//TASK 5-2 enable light rotation
rotateLight2.matrix = glm.rotateY(-timeInMilliseconds*0.1);

root.render(context);
```

# Extra Task 5: Shader Solution

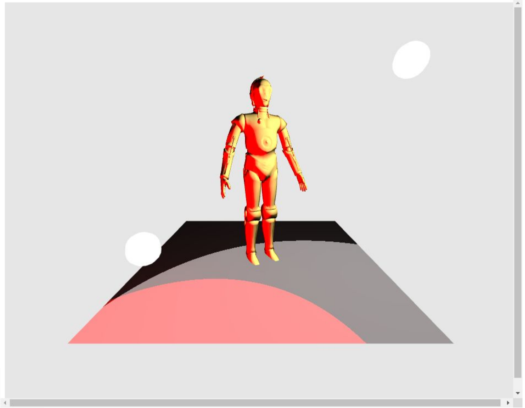
## Vertex shader:



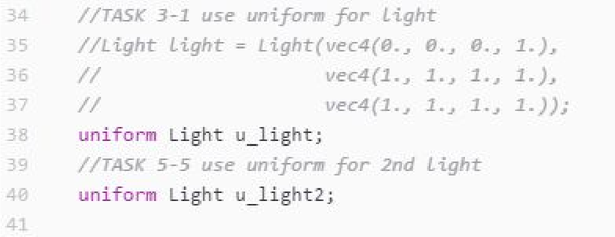
```
//TASK 3-3 light position as uniform
//vec3 lightPos = vec3(0, -2, 2);
uniform vec3 u_lightPos;
//TASK 5-3 second light source
uniform vec3 u_light2Pos;

```



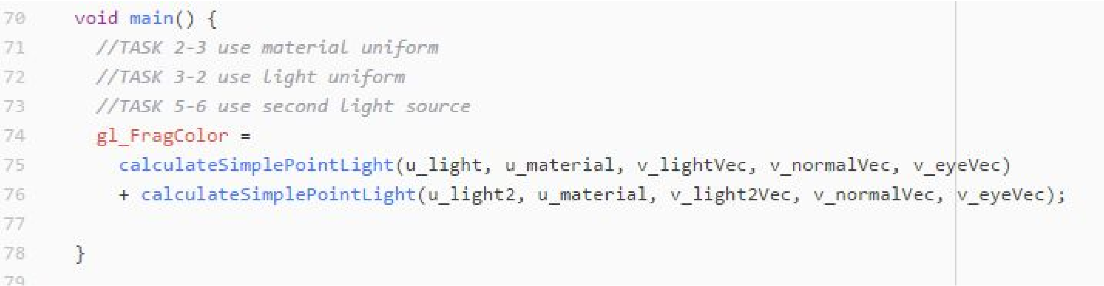
```
  v_eyeVec = -eyePosition.xyz;
  //TASK 3-4 light position as uniform
  v_lightVec = u_lightPos - eyePosition.xyz;
  //TASK 5-4 second light source position
  v_light2Vec = u_light2Pos - eyePosition.xyz;

  gl_Position = u_projection * eyePosition;
}
```



## Fragment shader:



```
//TASK 3-1 use uniform for light
//Light light = Light(vec4(0., 0., 0., 1.),
//                    vec4(1., 1., 1., 1.),
//                    vec4(1., 1., 1., 1.));
uniform Light u_light;
//TASK 5-5 use uniform for 2nd light
uniform Light u_light2;

```

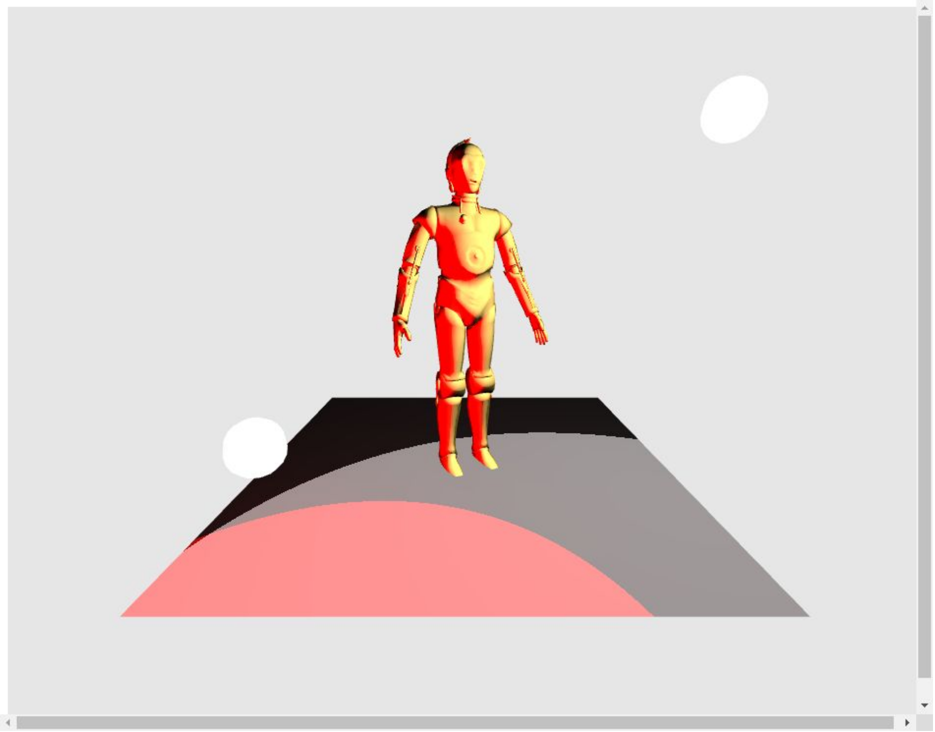


```
void main() {
  //TASK 2-3 use material uniform
  //TASK 3-2 use light uniform
  //TASK 5-6 use second light source
  gl_FragColor =
    calculateSimplePointLight(u_light, u_material, v_lightVec, v_normalVec, v_eyeVec)
    + calculateSimplePointLight(u_light2, u_material, v_light2Vec, v_normalVec, v_eyeVec);

}
```

# Recap

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

## Texturing

*How to map an image on a mesh?*