Aufgabe 1
Lighting in OpenGL

Steps:
1. Create light sources
2. Position light sources
3. Enable light sources
4. Define material for objects in the scene
Creating light sources

```java
float light_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
float light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

floatBuffer light_ambient() { 0.2f, 0.2f, 0.2f, 1.0f };
floatBuffer light_diffuse() { 1.0f, 1.0f, 1.0f, 1.0f };
floatBuffer light_specular() { 1.0f, 1.0f, 1.0f, 1.0f };

gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, FloatBuffer.wrap(light_ambient));
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, FloatBuffer.wrap(light_diffuse));
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPECULAR, FloatBuffer.wrap(light_specular));
```

Default values:
- Ambient: (0.0, 0.0, 0.0, 1.0)
- Diffuse: (1.0, 1.0, 1.0, 1.0)
- Specular: (1.0, 1.0, 1.0, 1.0)

Max. number of lights in a scene:
- At least 8 (GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, GL_LIGHT3, etc...
float light_position[] = { -10f, -6f, 10.0f, 0.0f };  
float light_direction[] = { 10f, 5f, -10.0f, 0.0f };  

gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION,  
FloatBuffer.wrap(light_position));  

gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPOT_DIRECTION,  
FloatBuffer.wrap(light_direction));  

Default values:  
• Position: (0.0, 0.0, 1.0, 0.0)  
• Direction: (0.0, 0.0, -1.0)
Enabling light sources

gl.glEnable(GL2.GL_LIGHTING);
gl.glEnable(GL2.GL_LIGHT0);
## Further Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_AMBIENT</td>
<td>(0.0, 0.0, 0.0, 1.0)</td>
<td>ambient RGBA intensity of light</td>
</tr>
<tr>
<td>GL_DIFFUSE</td>
<td>(1.0, 1.0, 1.0, 1.0)</td>
<td>diffuse RGBA intensity of light</td>
</tr>
<tr>
<td>GL_SPECULAR</td>
<td>(1.0, 1.0, 1.0, 1.0)</td>
<td>specular RGBA intensity of light</td>
</tr>
<tr>
<td>GL_POSITION</td>
<td>(0.0, 0.0, 1.0, 0.0)</td>
<td>(x, y, z, w) position of light</td>
</tr>
<tr>
<td>GL_SPOT_DIRECTION</td>
<td>(0.0, 0.0, -1.0)</td>
<td>(x, y, z) direction of spotlight</td>
</tr>
<tr>
<td>GL_SPOT_EXPONENT</td>
<td>0.0</td>
<td>spotlight exponent</td>
</tr>
<tr>
<td>GL_SPOT_CUTOFF</td>
<td>180.0</td>
<td>spotlight cutoff angle</td>
</tr>
<tr>
<td>GL_CONSTANT_ATTENUATION</td>
<td>1.0</td>
<td>constant attenuation factor</td>
</tr>
<tr>
<td>GL_LINEAR_ATTENUATION</td>
<td>0.0</td>
<td>linear attenuation factor</td>
</tr>
<tr>
<td>GL_QUADRATIC_ATTENUATION</td>
<td>0.0</td>
<td>quadratic attenuation factor</td>
</tr>
</tbody>
</table>

http://www.glprogramming.com/red/chapter05.html
Defining Materials

```java
float no_mat[] = { 0.0f, 0.1f, 0.1f, 1.0f };
float mat_ambient[] = { 1f, 1f, 1f, 1.0f };
float mat_diffuse[] = { 0.1f, 0.5f, 0.8f, 1.0f };
float mat_diffuse2[] = { 0.8f, 0.8f, 0.8f, 1.0f };
float mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float no_shininess[] = { 0.0f };
float low_shininess[] = { 80.0f };
float mat_emission[] = {0.3f, 0.2f, 0.2f, 0.0f};

gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT, FloatBuffer.wrap(mat_ambient));
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, FloatBuffer.wrap(mat_diffuse));
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SPECULAR, FloatBuffer.wrap(mat_specular));
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_SHININESS, FloatBuffer.wrap(low_shininess));
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_EMISSION, FloatBuffer.wrap(no_mat));
```
Applying Textures

• JOGL provides some utility classes

Steps:
1. Specify texture coordinates
2. Load a texture
3. Enable and bind a texture
Texture coordinates

```cpp
gl.glBegin(GL2.GL_QUADS);

// Front face of a cube
gl.glNormal3d(0, 0, 1);
gl.glTexCoord2d(0, 0);
gl.glVertex3d(0, 0, 0);
gl.glTexCoord2d(1, 0);
gl.glVertex3d(1, 0, 0);
gl.glTexCoord2d(1, 1);
gl.glVertex3d(1, 1, 0);
gl.glTexCoord2d(0, 1);
gl.glVertex3d(0, 1, 0);
```
Applying Textures

```java
private Texture ground; // texture variable
...
// loading from image file, can be done in init()-method
try{
    InputStream stream = getClass().getResourceAsStream("grass.png");
    TextureData data = TextureIO.newTextureData(gl.getGLProfile(),stream, false,"png");
    ground = TextureIO.newTexture(data);
}

// enabling and binding (JOGL-specific, in display-method()
ground.enable(gl);
ground.bind(gl);
// draw objects with texture applied here
ground.disable(gl);
```

For good performance: Image files should be square, with dimensions as powers of 2.
Aufgabe 2
Terrain - What did we have so far?

Polygon mesh + varying color based on height
Triangle strips drawn in wireframe mode
No light
Now: Light + surface description
Light

Basic light setup:

```java
float light_ambient[] = { 0.1f, 0.1f, 0.1f, 1.0f};
float light_diffuse[] = { 1.0f, 1.0f, 0.8f, 1.0f};
float light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
float light_position[] = { 40.0f, 55.0f, 0.0f, 0.0f};
float light_direction[] = { -40.0f, -55.0f, 0.0f, 0.0f};

gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, FloatBuffer.wrap(light_ambient));
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, FloatBuffer.wrap(light_diffuse));
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPECULAR, FloatBuffer.wrap(light_specular));
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION, FloatBuffer.wrap(light_position));
gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPOT_DIRECTION, FloatBuffer.wrap(light_direction));
gl.glLightf(GL2.GL_LIGHT0, GL2.GL_SPOT_CUTOFF, 45.0f);

gl.glEnable(GL2.GL_LIGHTING);
gl.glEnable(GL2.GL_LIGHT0);
```
Material

Basic material setup:

```java
float no_mat[] = { 0.0f, 0.0f, 0.0f, 1.0f };
float mat_amb[] = { 0.2f, 0.2f, 0.2f, 1.0f };
float mat_diffuse[] = { 0.1f, 0.8f, 0.2f, 1.0f };

gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT, FloatBuffer.wrap(mat_amb));
gl.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, FloatBuffer.wrap(mat_diffuse));
```
gl.glClearColor(0.8f, 0.9f, 0.9f, 0f); // sky-color background

Now: Light, but we can only see a silhouette. Why?
We need do calculate surface normals

- Let’s think about the polygon mesh

We can calculate one normal for every triangle in the strip (flat shading):
  - Cross product of two in-plane vectors
  - Problem: Normals need to be defined per vertex in OpenGL
  - In a triangle strip vertices are shared – in our example we have 200 vertices and 198 triangles per strip.
Calculation of face normals

\[ \vec{ip}_1 = \vec{b} - \vec{a} \]
\[ \vec{ip}_2 = \vec{c} - \vec{a} \]
\[ \vec{ip}_3 = \vec{d} - \vec{c} \]

Face normals:
\[ \vec{ip}_1 \times \vec{ip}_2 \]
\[ \vec{ip}_3 \times \vec{ip}_2 \]

In total:
99 * 198 = 19602 face normals
Idea: Average normals

- Let’s think more about the polygon mesh

- Calculate face normals in a first step
- Based on the face normals calculate the vertex normals
- Enables "smooth" shading

Vertex normal = sum(face normals)/count(face normals)
The calculation is not difficult, but the number of adjacent faces differs.
Result
Texture Overlay