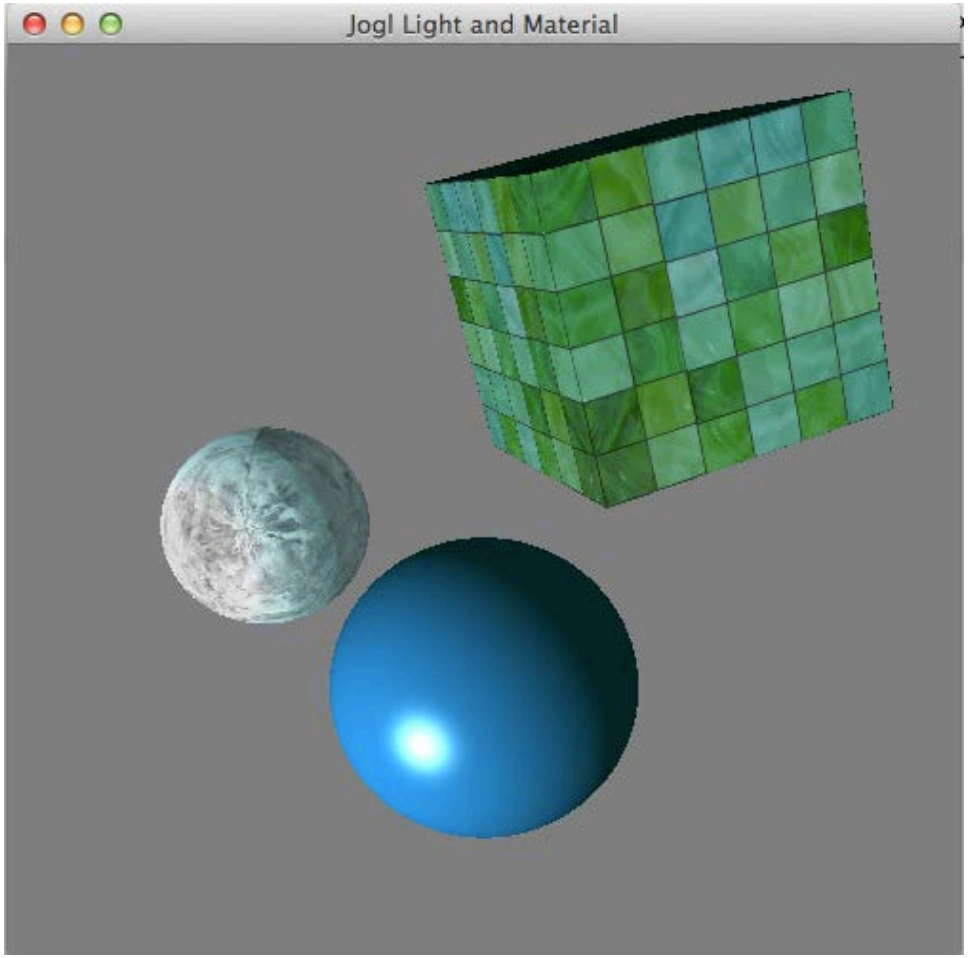
Computergrafik 1

Blatt 8

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

```
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 };

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…)

Position of light sources

```
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

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

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

Defining Materials

```
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

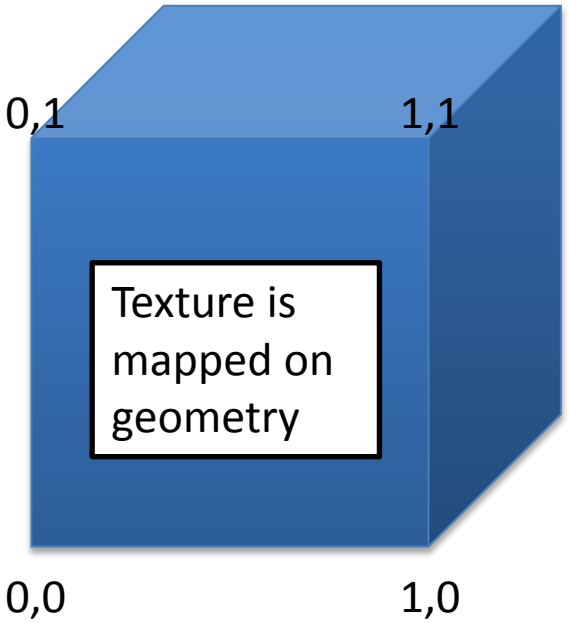
- JOGL provides some utility classes

Steps:

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

Texture coordinates

```
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);
```

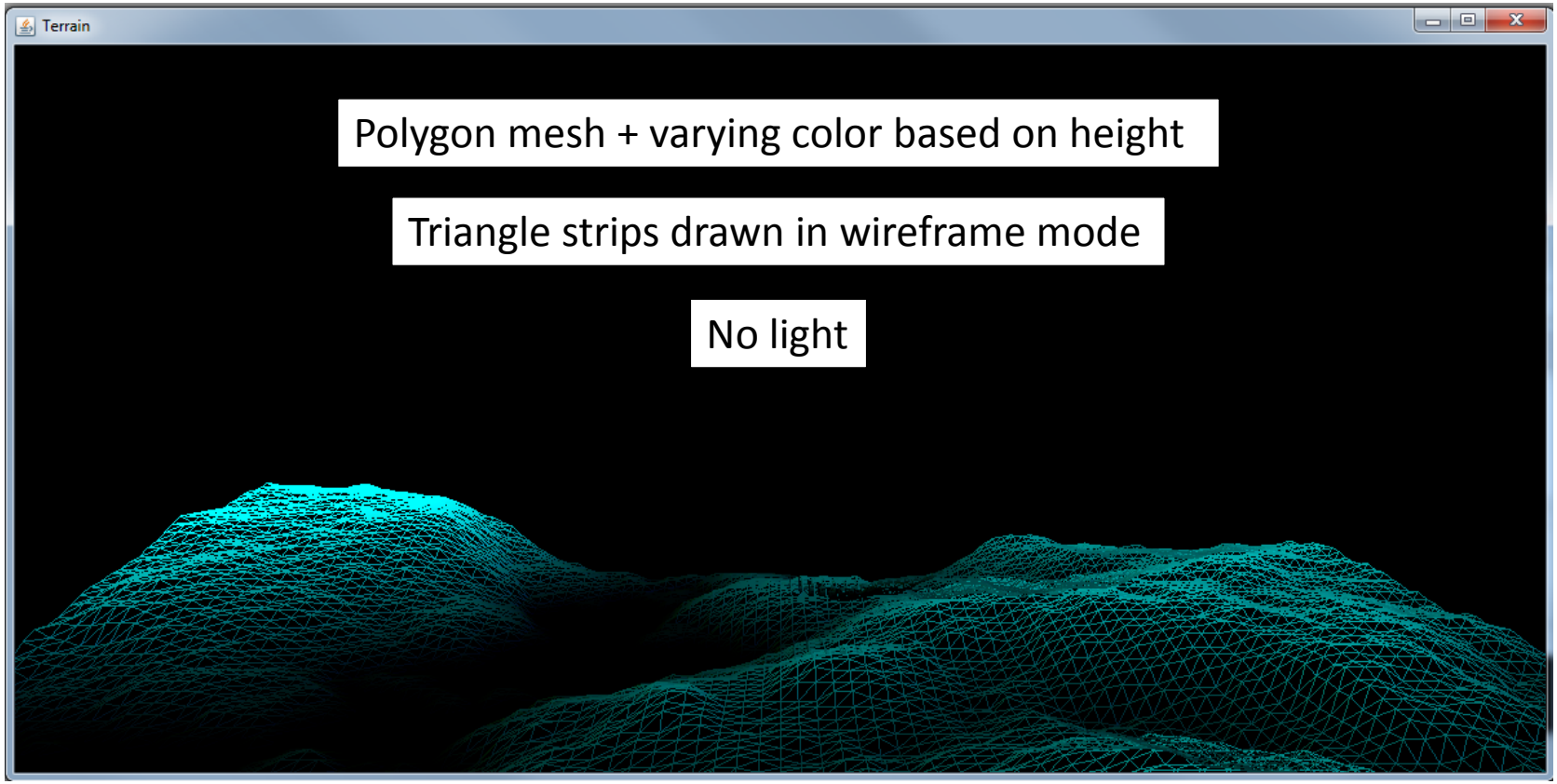


Applying Textures

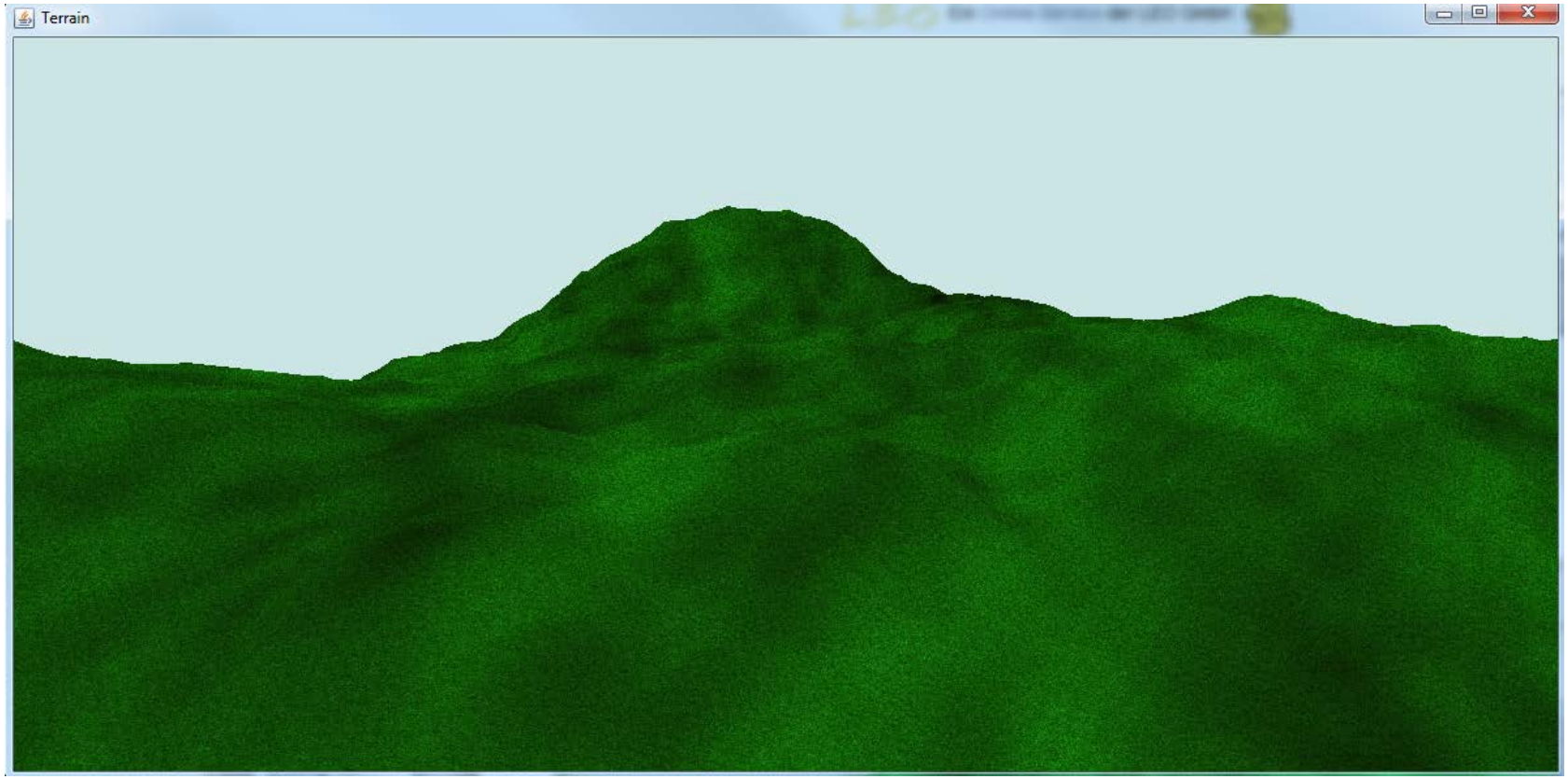
```
private Texture ground; // texture variable
…
//loading from image file, can be done in init()-method
try{
InputStream stream = getClass().getResourceAsStream("grass.png");
TextureData data =  TextureIO.newTextureData(gl.getGLProfile(),stream, false,"png");

ground = TextureIO.newTexture(data);

} catch(IOException e){
          e.printStackTrace();
}
....
//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?



Terrain – Part 2



Now: Light + surface description

Light

Basic light setup:

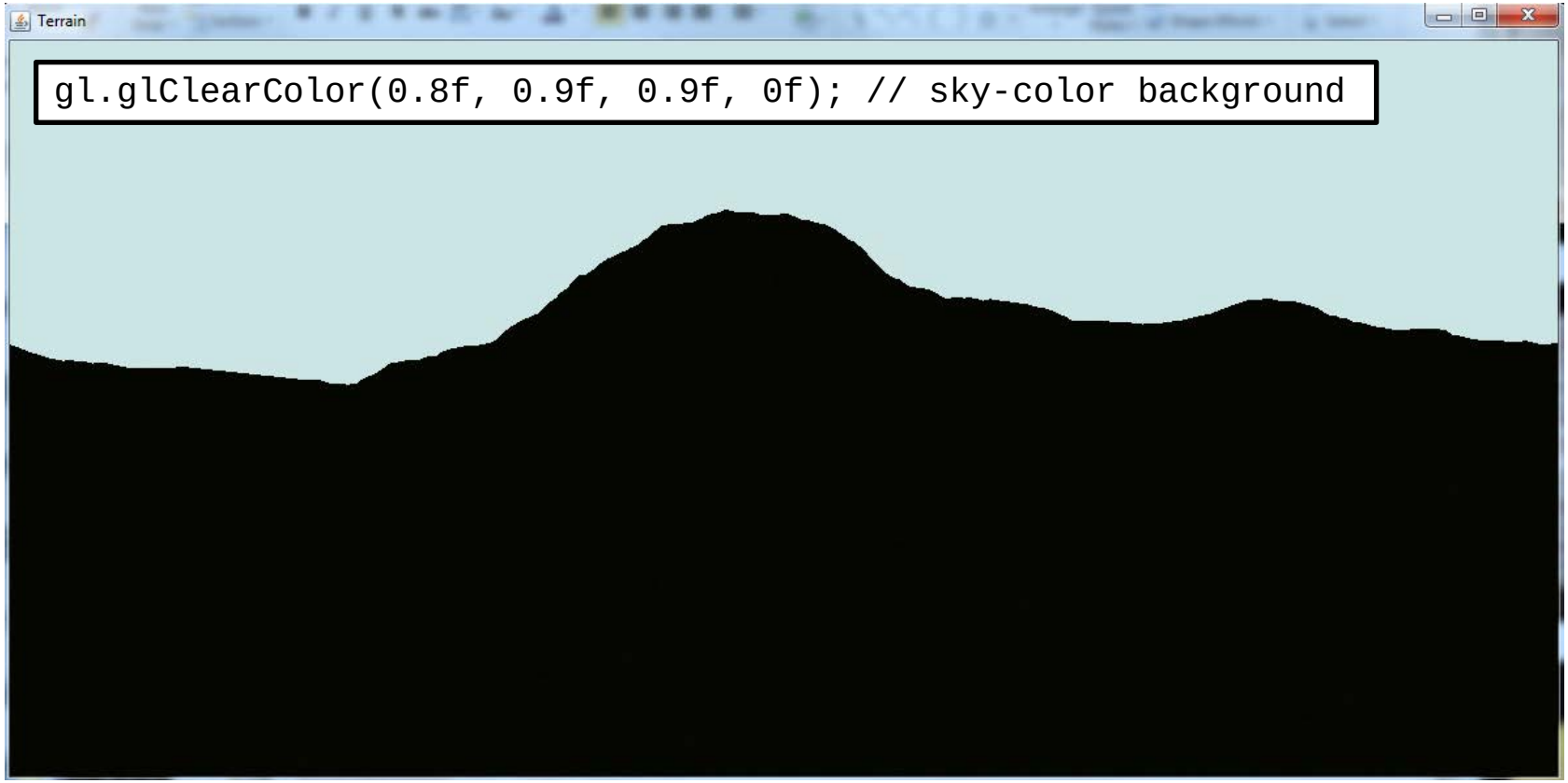
```
float light_ambient[] = { 0.1f, 0.1f, 0.1f, 1.0f };
float light_diffuse[] = { 1.0f, 1.0f, 0.8f, 1.0f };
float light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float light_position[] = { 40.0f, 55.0f, 0.0f, 0.0f };
float light_direction[] = { -40.0f, -55.0f, 0.0f, 0.0f };

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:

```
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));
```

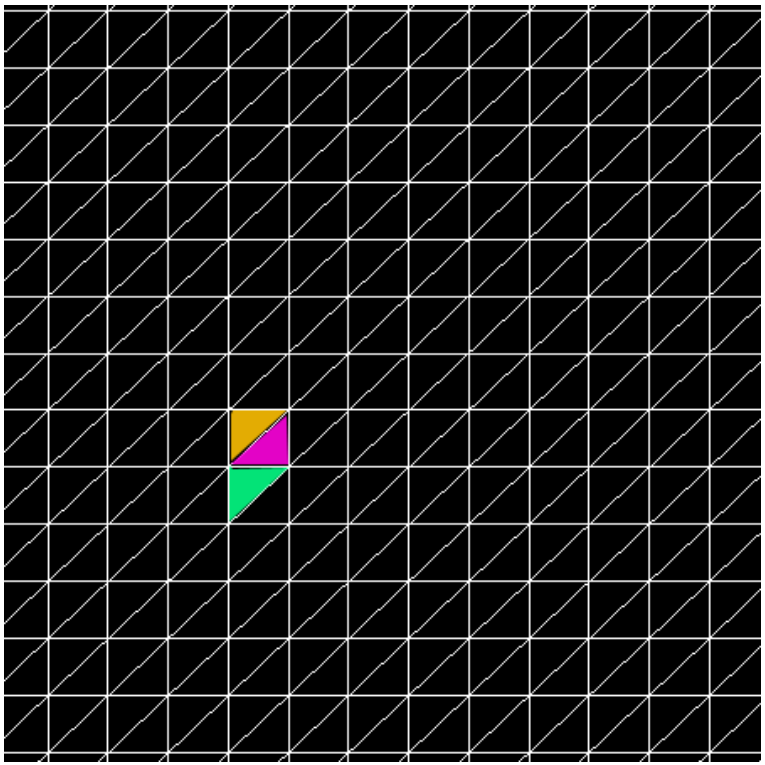
Result



Now: Light, but we can only see a silhouette. Why?

We need do calculate surface normals

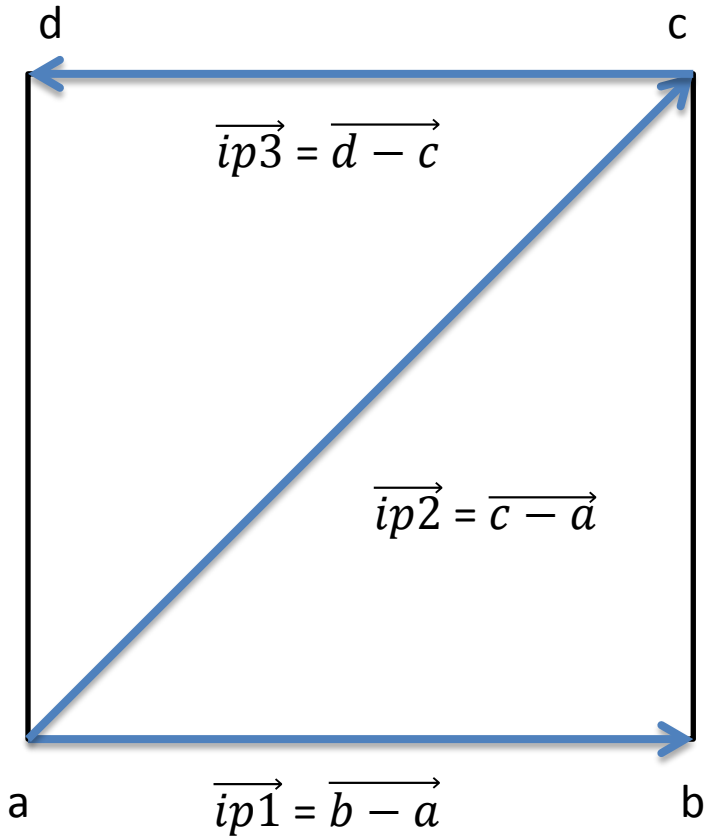
- Let‘s think about the polygon mesh



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



Face normals:

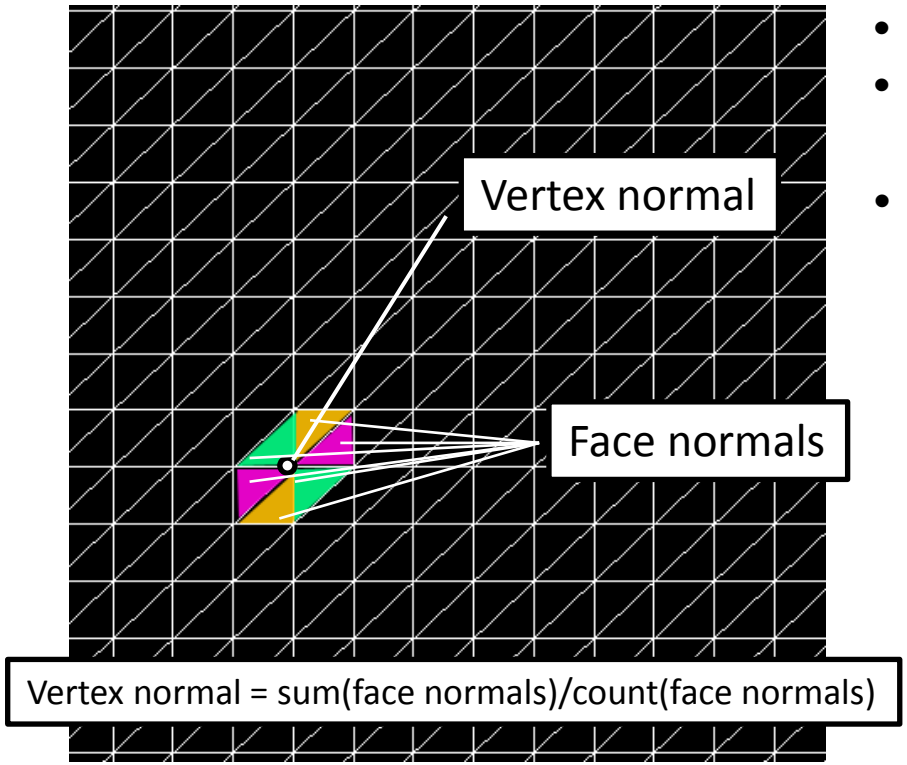
$\overrightarrow{ip1} \times \overrightarrow{ip2}$

$\overrightarrow{ip3} \times \overrightarrow{ip2}$

In total:
99 * 198 = 19602 face normals

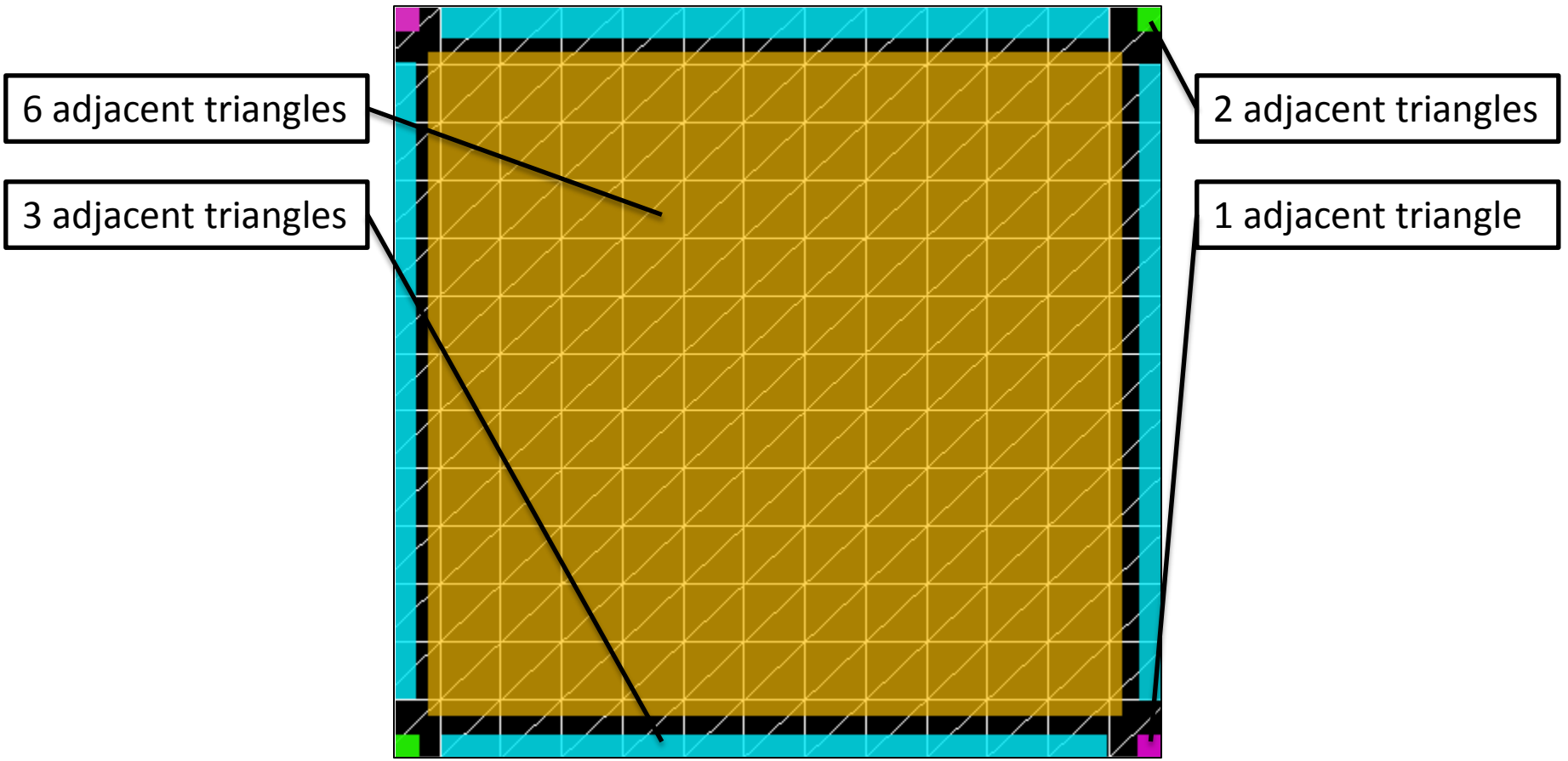
Idea: Average normals

- Let‘s think more about the polygon mesh



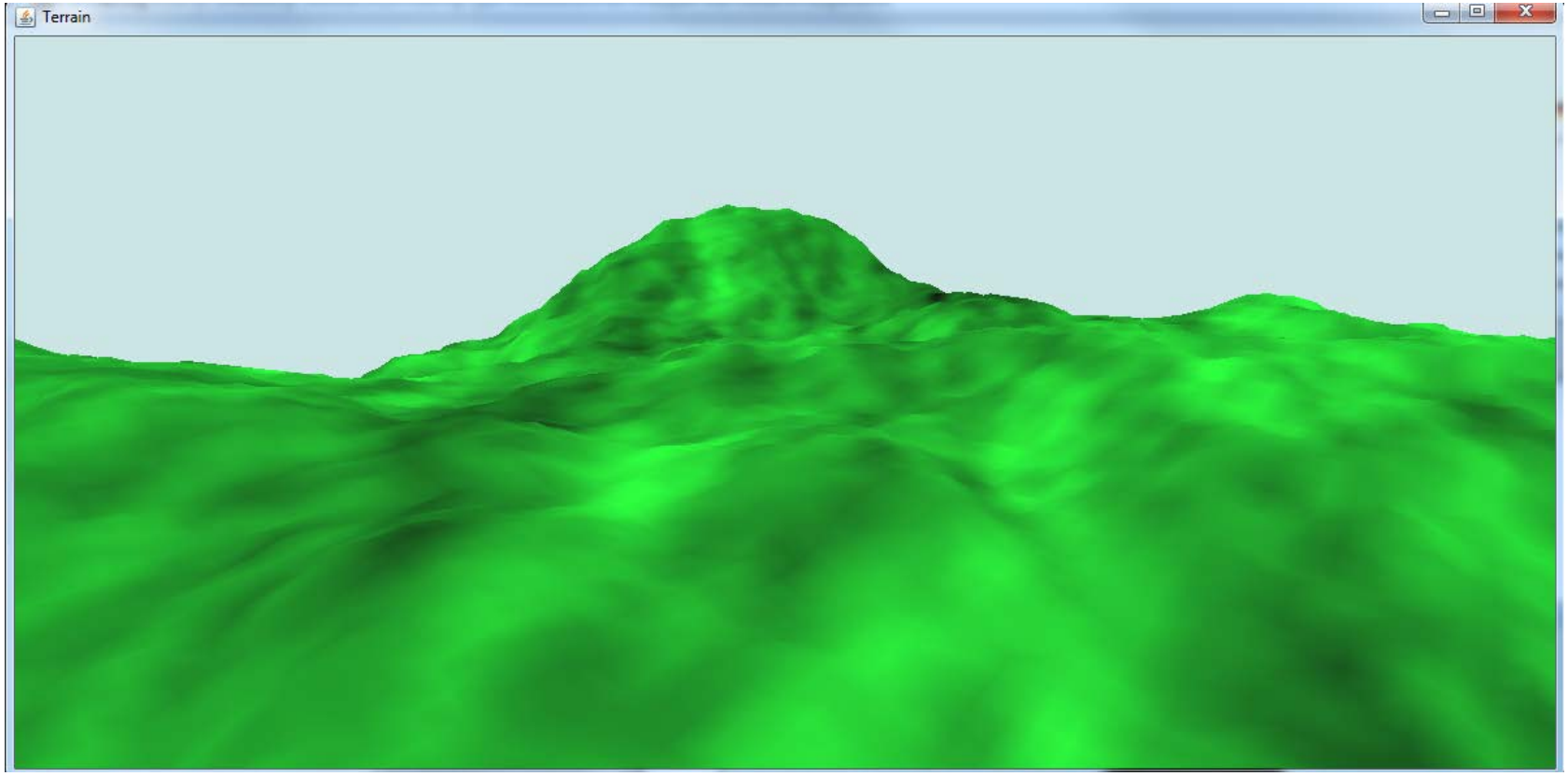
- Calculate face normals in a first step
- Based on the face normals calculate the vertex normals
- Enables „smooth“ shading

Calculation of vertex normals



The calculation is not difficult, but the number of adjacent faces differs

Result



Texture Overlay