Modernes OpenGL

Lorenz Diener

Entropia EV

19. Juni 2010

Einführung

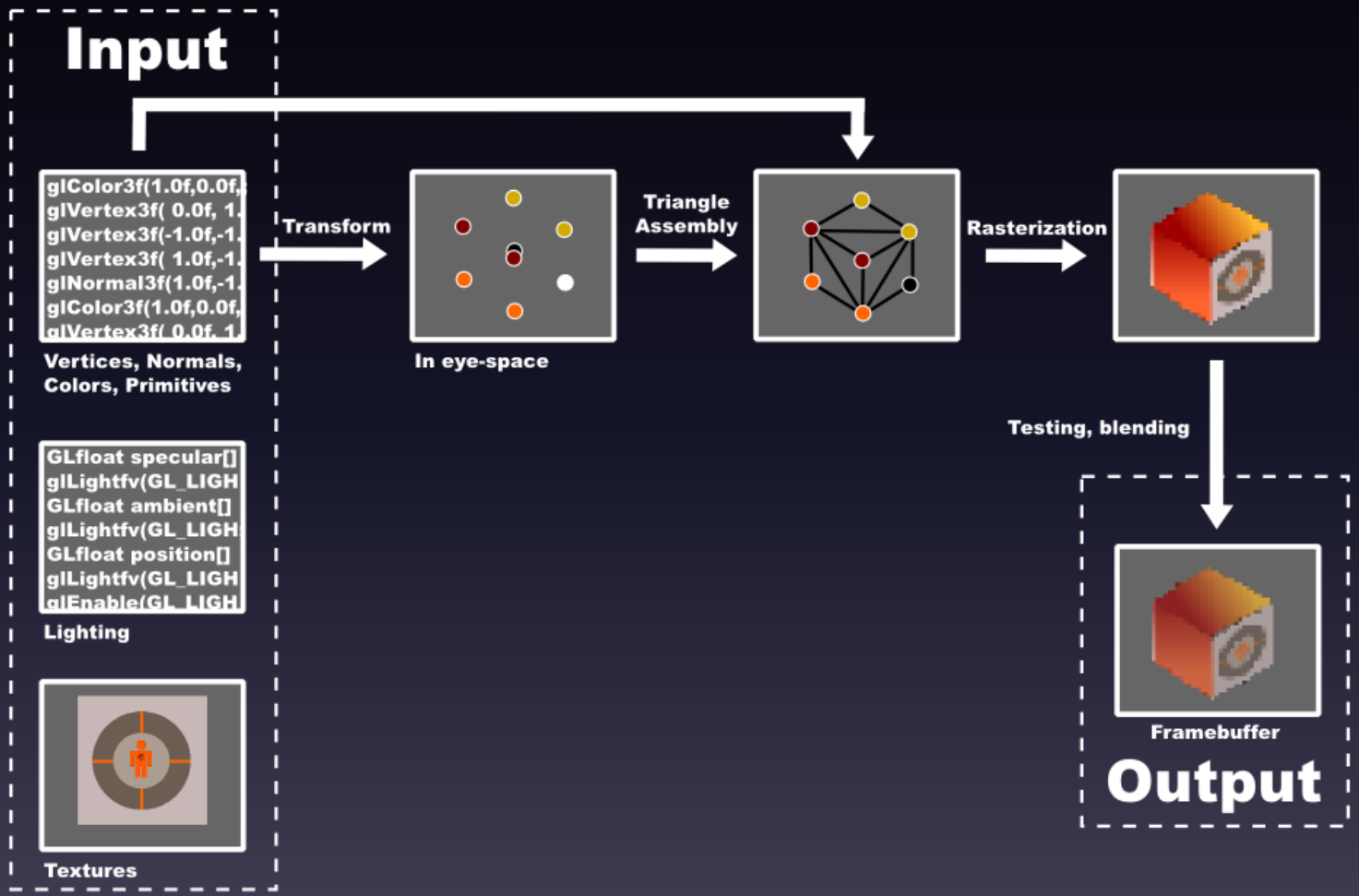
Einführung

- Aktuelle Version: 4.0
- Dokumentation nicht sehr Anfängerfreundlich
- Schwierig zu sehen, was aktueller Stand ist

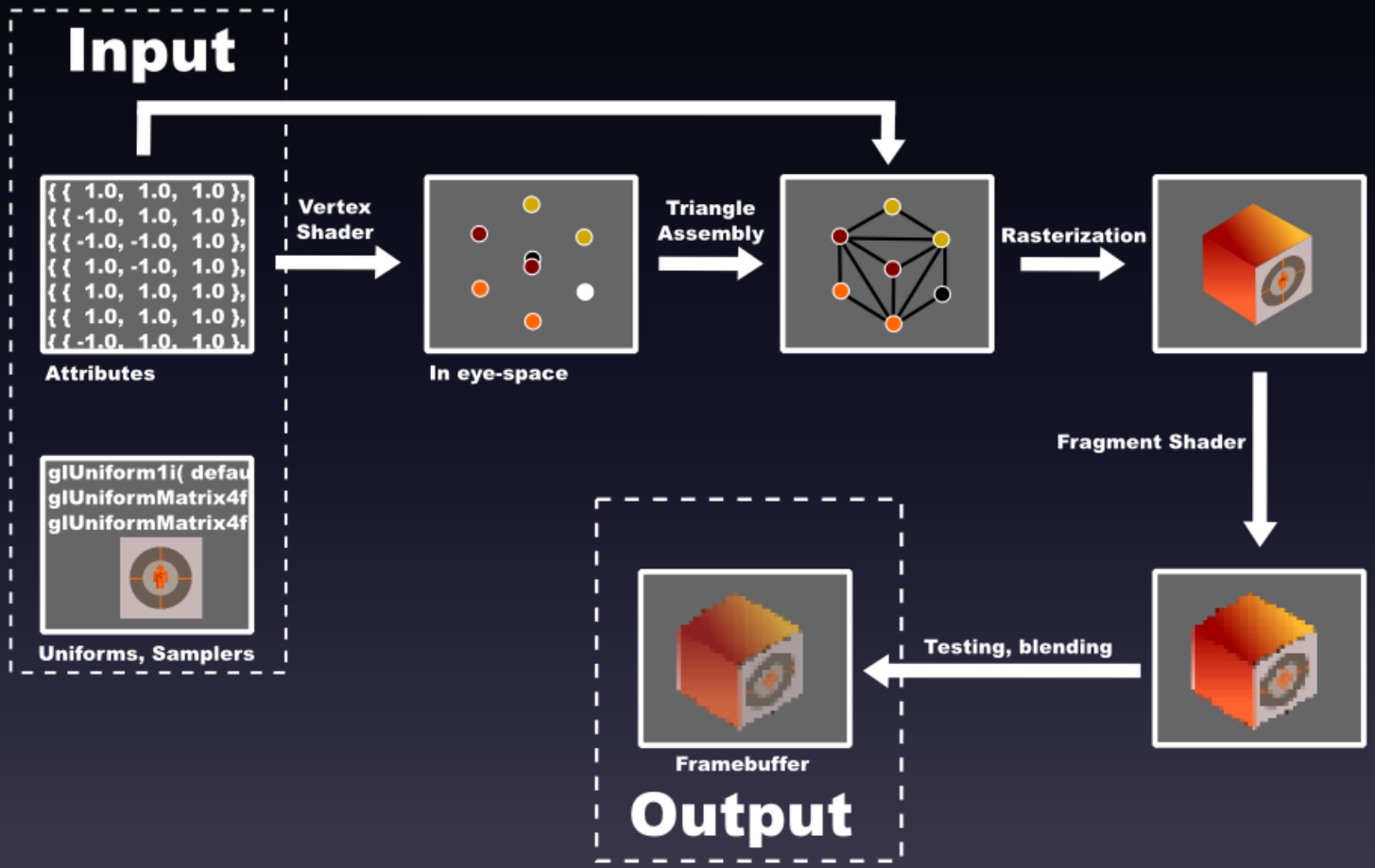
Einführung

- Heute: Einführung
- OpenGL in modern
- Auch WebGL, OpenGL ES 2.0
- Demo-Programm (GLUT, GLEW)

Fixed-Function-Pipeline



Programmable Pipeline



Ein Fenster Aufmachen

```
glutInit( &argc, argv );
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );

char modeString[255] = "";
sprintf( modeString, "%dx%d:24",
      WINDOW_WIDTH, WINDOW_HEIGHT );
glutGameModeString( modeString );
glutEnterGameMode();

glewInit();

glutDisplayFunc( draw );
glutTimerFunc( 15, update, 0 );
```

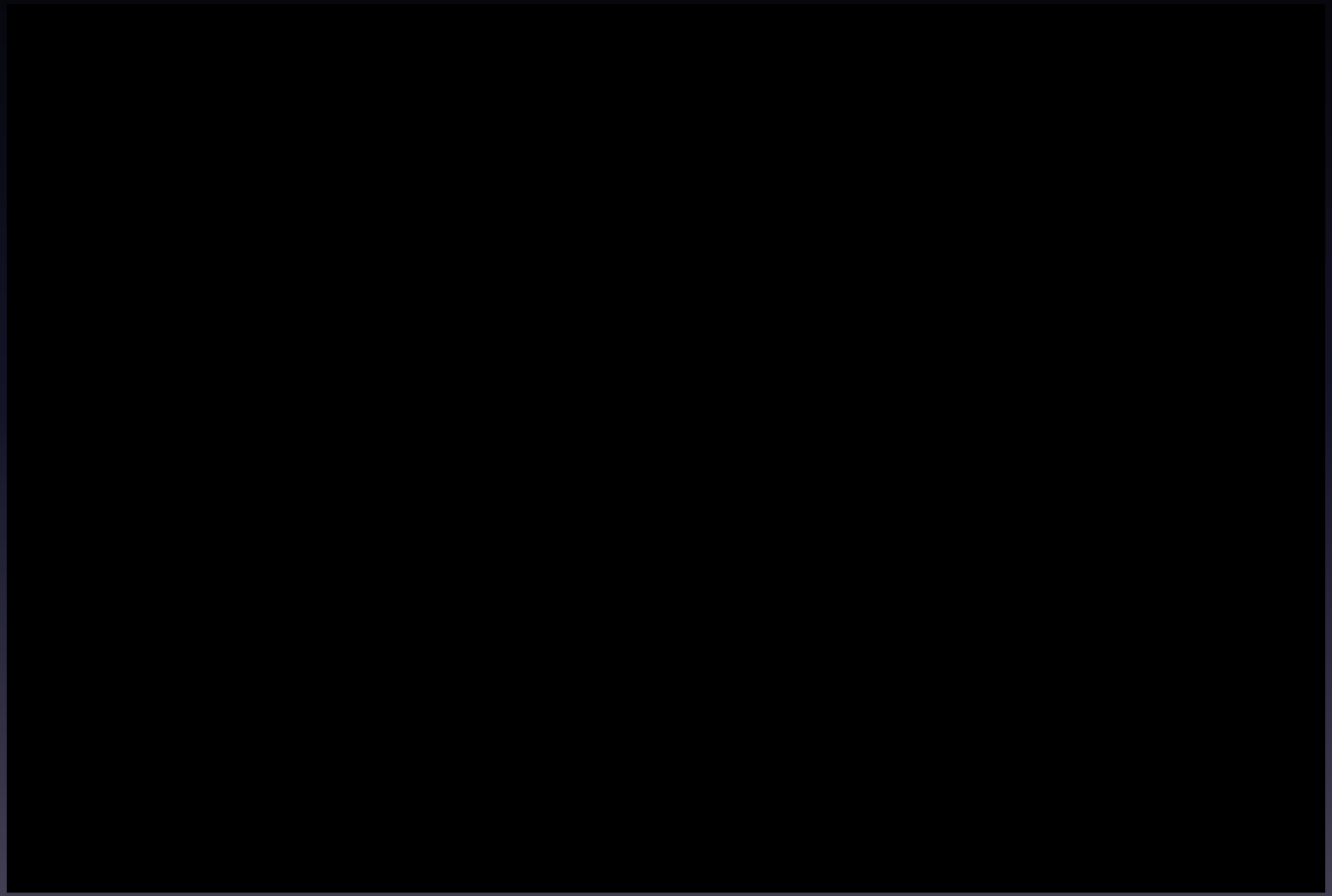
OpenGL-Setup

```
glEnable( GL_DEPTH_TEST );
glClearDepth( 1.0f );
glDepthFunc( GL_LEQUAL );

glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );

glEnable( GL_CULL_FACE );
glCullFace( GL_BACK );
```

Ergebnis



Buffer-Objects

```
vertex vertices[] = {
{ { 1.0, 1.0, 1.0 }, 1.0,
{ 0.0, 0.0, 1.0 },
0.0, 0.0 },

{ { -1.0, 1.0, 1.0 }, 1.0,
{ 0.0, 0.0, 1.0 },
1.0, 0.0 }

[...]
```

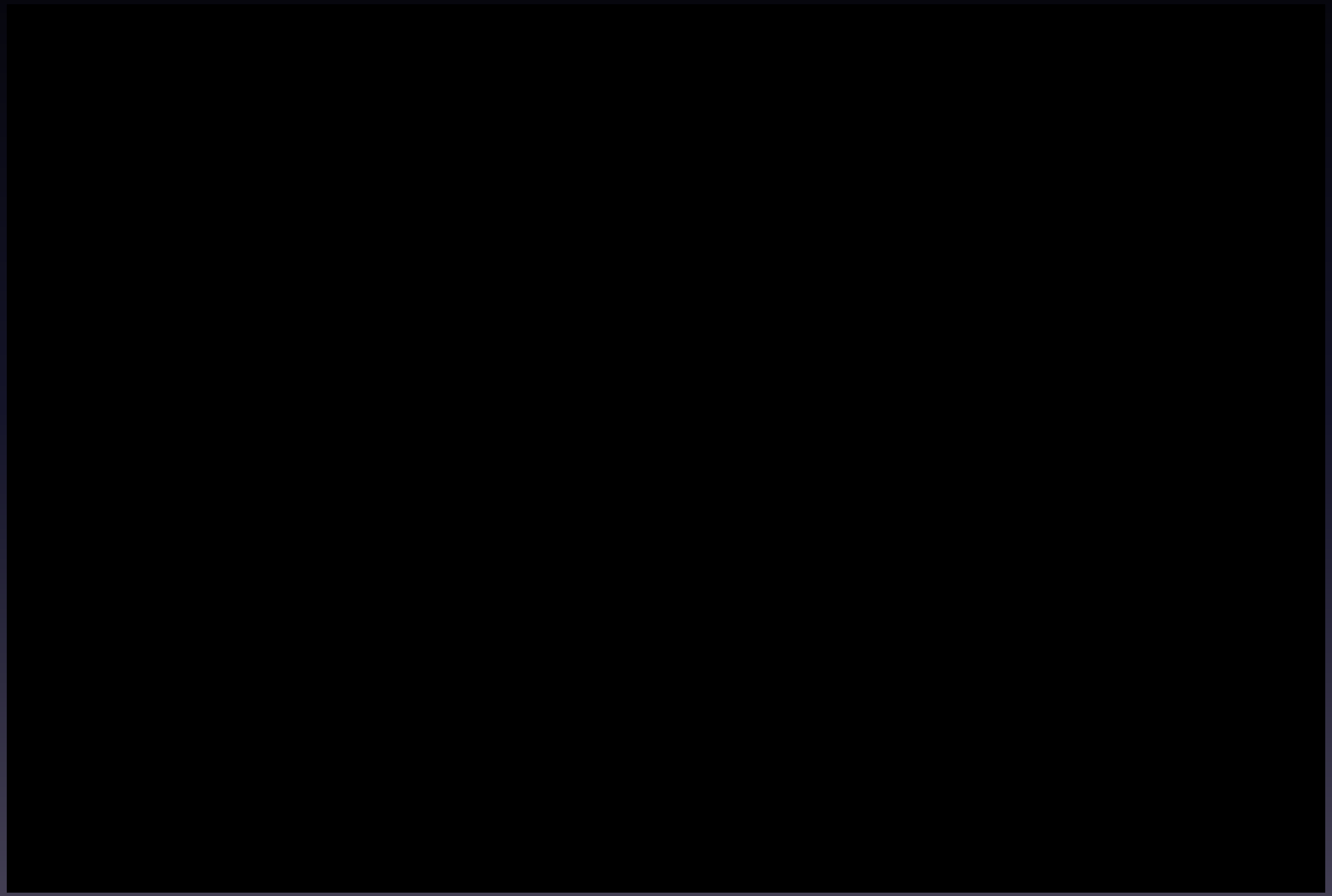
Buffer-Objects

```
GLuint bo;
glGenBuffers( 1, &bo );
glBindBuffer( GL_ARRAY_BUFFER, bo );
glBufferData(
    GL_ARRAY_BUFFER,
    vertices,
    sizeof( vertices ),
    GL_STATIC_DRAW
);
```

Texturen

- Laden: Library, selbstgemachter loader
- glGenTextures
- glBindTexture
- glTexParameteri - Interpolation!
- glTexImage2D

Ergebnis



Shader

- Vertex Shader, Fragment Shader, (Geometry Shader)
- Massiv parallel
- GLSL

Vertex-Shader

- Eingabe: Attribute
- Ausgabe: gl_Position, varying-variablen
- gl_Position = viewVertex * modelview * projection;

Modelview-Matrix

- Lineare Transformationen: Rotation, Skalierung, Scherung
- "Gleiche Werte, andere Achsen."
- http://en.wikipedia.org/wiki/Linear_map

Homogene Koordinaten

- Verschiebung ist nicht linear
- Idee: Eine weitere Reihe in der Matrix, eine 1.0 am ende jedes Vektors
- (Später auch noch mal nützlich)

Projection-Matrix

- Projection space
- Würfel von -1 bis 1, alles ausserhalb: Clipping
- Vorderfläche entspricht Bildschirm
- Z-Achse zeigt in den Bildschirm
- World-space hierauf abbilden - Perspektivprojektion.

Fragment-Shader

- Eingabe: Varyings vom Vertex-Shader
- Ausgabe: gl_FragColor
- gl_FragColor = vec4(1.0, 0.0, 1.0, 1.0);

Shader laden

```
GLchar *shaderSrc = loadFile( file );
GLsizei length = strlen ( shaderSrc );
GLuint shader = glCreateShader( type );
glShaderSource(
    shader,
    1,
    (const GLchar**)&shaderSrc,
    &length
);
free( shaderSrc );
glCompileShader( shader );
```

Shader linken

```
GLuint vertexShader =
    loadShader( GL_VERTEX_SHADER, "simple.vert" );
GLuint fragmentShader =
    loadShader( GL_FRAGMENT_SHADER, "simple.frag" );
shaderProgram =
    makeShaderProgram( vertexShader, fragmentShader );
vertexNormal =
    glGetAttribLocation( shaderProgram, "position" );
modelviewMatrix =
    glGetUniformLocation( shaderProgram, "modelview" );
projectionMatrix =
    glGetUniformLocation( shaderProgram, "projection" );
```

Rendern, Teil 1

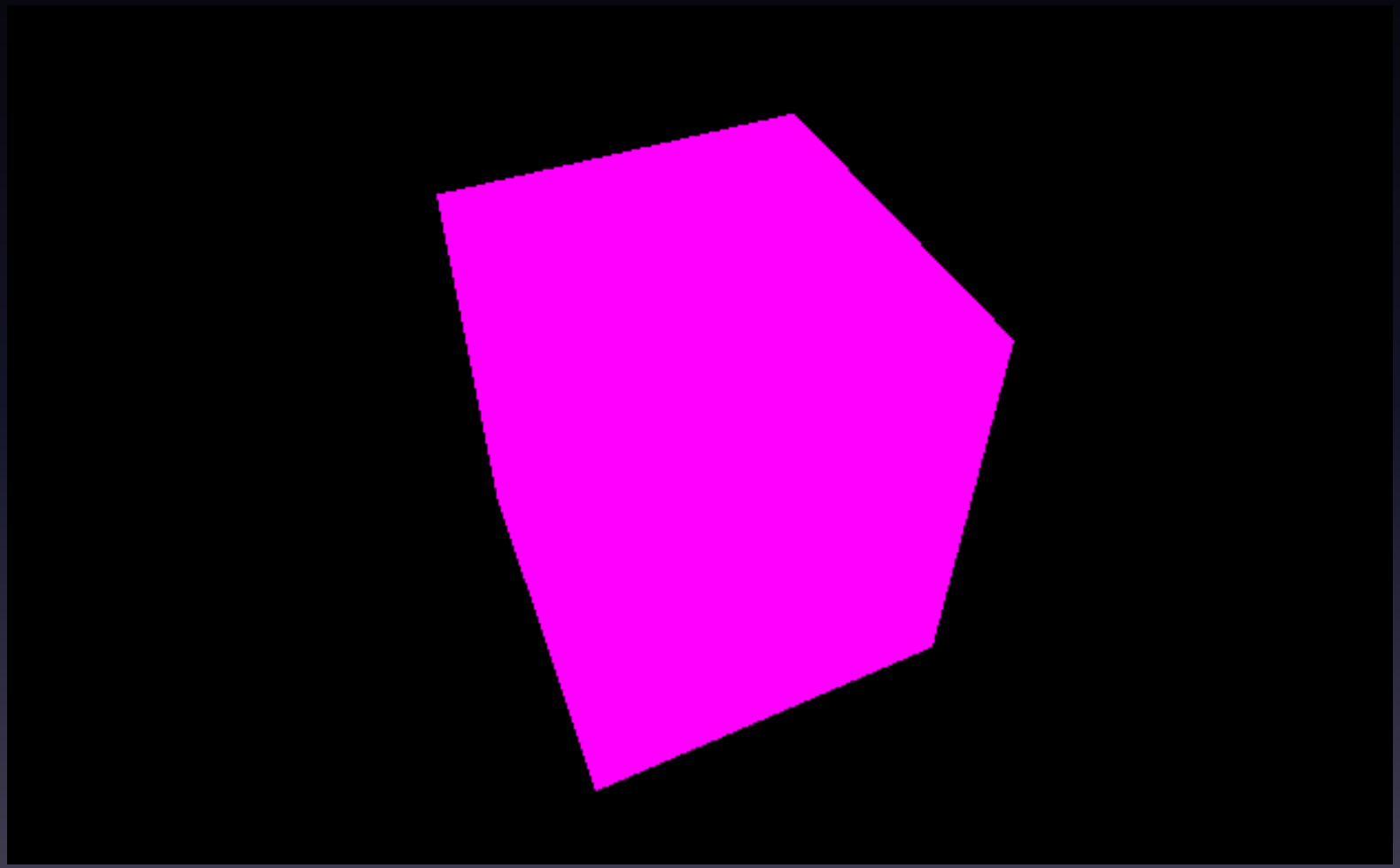
```
glUseProgram( shaderProgram );
glUniformMatrix4fv( modelviewMatrix, 1, GL_TRUE, mv );
glUniformMatrix4fv( projectionMatrix, 1, GL_TRUE, proj );
glBindBuffer( GL_ARRAY_BUFFER, vertexBuffer );
glVertexAttribPointer(
    vertexPosition,
    4,
    GL_FLOAT,
    GL_FALSE,
    sizeof(GLfloat) * 9,
    (void*)0
);
glEnableVertexAttribArray( vertexPosition );
```

Rendern, Teil 2

```
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, elementBuffer );
glDrawElements(
    GL_QUADS,
    24,
    GL_UNSIGNED_SHORT,
    (void*)0
);

glutSwapBuffers();
```

Ergebnis

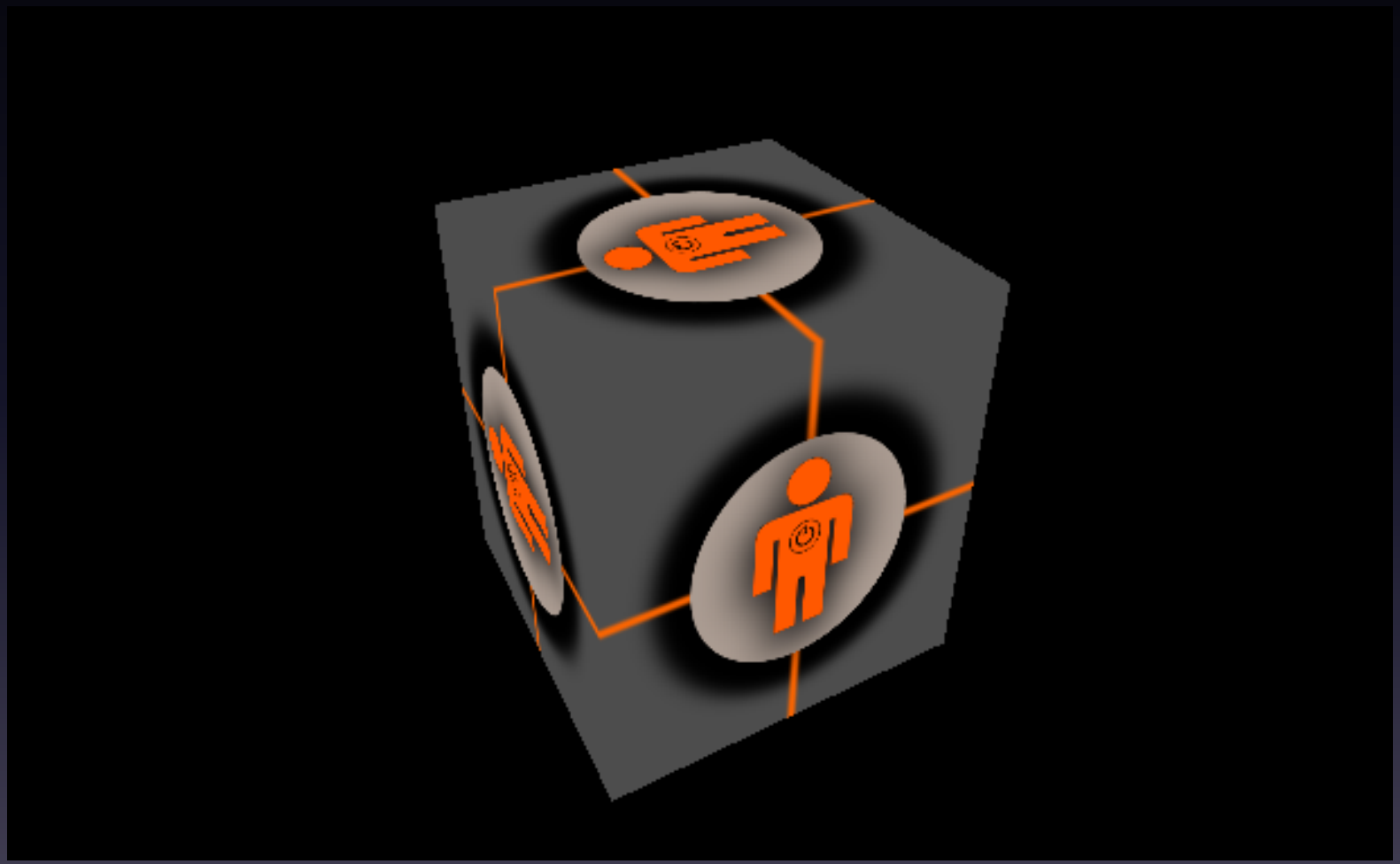


Texturieren

- An texture-unit binden
- Texturkoordinaten als attribut
- Per varying an Fragment-Shader - Interpolation!
- Im Fragment Shader: texture2D(texture, texcoord);

```
glActiveTexture( GL_TEXTURE0 );
glBindTexture( GL_TEXTURE_2D, defaultTextureData );
glUniform1i( defaultTexture, 0 );
```

Ergebnis



Licht

- Einfach: Phong-Shading
- Ambient, Diffuse, Specular
- http://en.wikipedia.org/wiki/Phong_shading
- Normalen (parameter) - auch transformiert!

```
viewVertex = vertex * modelview;
eye = normalize( viewVertex.xyz );
light = normalize( lightPos - viewVertex.xyz );
normal = normalize( (vec4(vnormal, 0.0) * normalview ).xyz );
```

Licht, Fragment-Shader

```
float lambertTerm = dot( normal, light );
if( lambertTerm > 0.0 ) {
    color += lightDiffuse * materialColor * lambertTerm;
    vec3 specDir = normalize( reflect( light, -normal ) );
    float specular = pow(
max( 0.0, dot(specDir, eye) ),
materialShinyness
    );
    color += lightSpecular *
    materialSpecular *
    specular;
}
```

Ergebnis

Code

Fragen?