OpenGL Einführung

Lehrstuhl für Computergraphik und
Multimediasysteme

Universität Siegen

Ziele dieser Einführung:

- funktionsfähige Arbeitsumgebung
- grundlegendes Verständnis von OpenGL

Arbeitsumgebung:

- CG Pool (dummy account)
- Ihr Computer/Notebook
- NVidia GPU
  - Intel und AMD problematisch

Voraussetzungen an die Arbeitsumgebung:

- Qt5
- OpenGL 4
  - in CG I: *core profile* von OpenGL 4
- CMake
- IDE + Compiler
  - Microsoft Visual Studio + cl
  - QtCreator + gcc/g++

Einrichten der Arbeitsumgebung

Code: helloworld.zip auf `http://www.cg.informatik.uni-siegen.de/de/computergraphik-i-2016w/uebungen`

Anleitung: `http://www.cg.informatik.uni-siegen.de/idehowto`

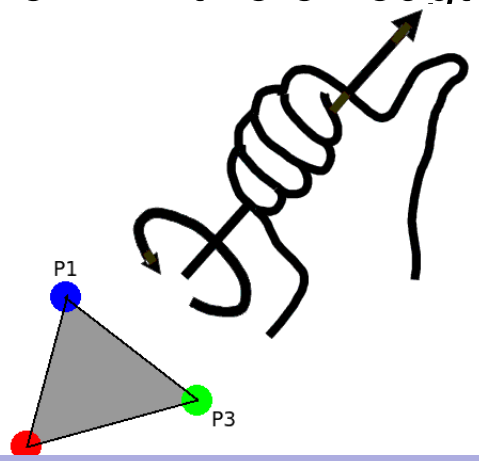
Baustein(e): Punkt/Punkte (engl: vertex/vertices)

Punkte:

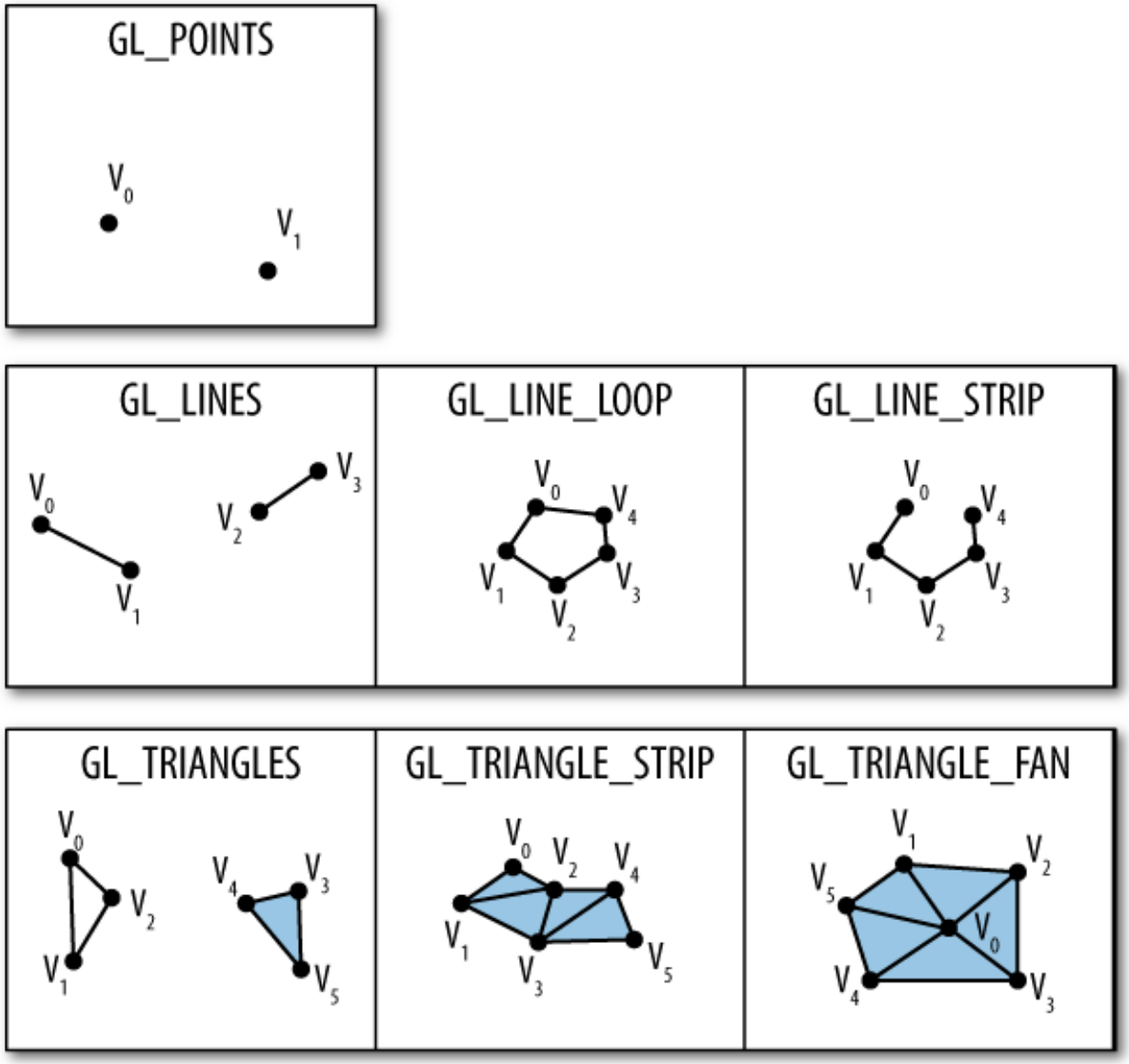
- 3D (euklidische Koordinaten) oder 4D (homogene Koordinaten)
- Eigenschaften:
  - Normalenvektor (3D oder 4D)
  - Farbe (3D oder 4D)
  - Texturkoordinaten (2D)

Geometrische Primitive

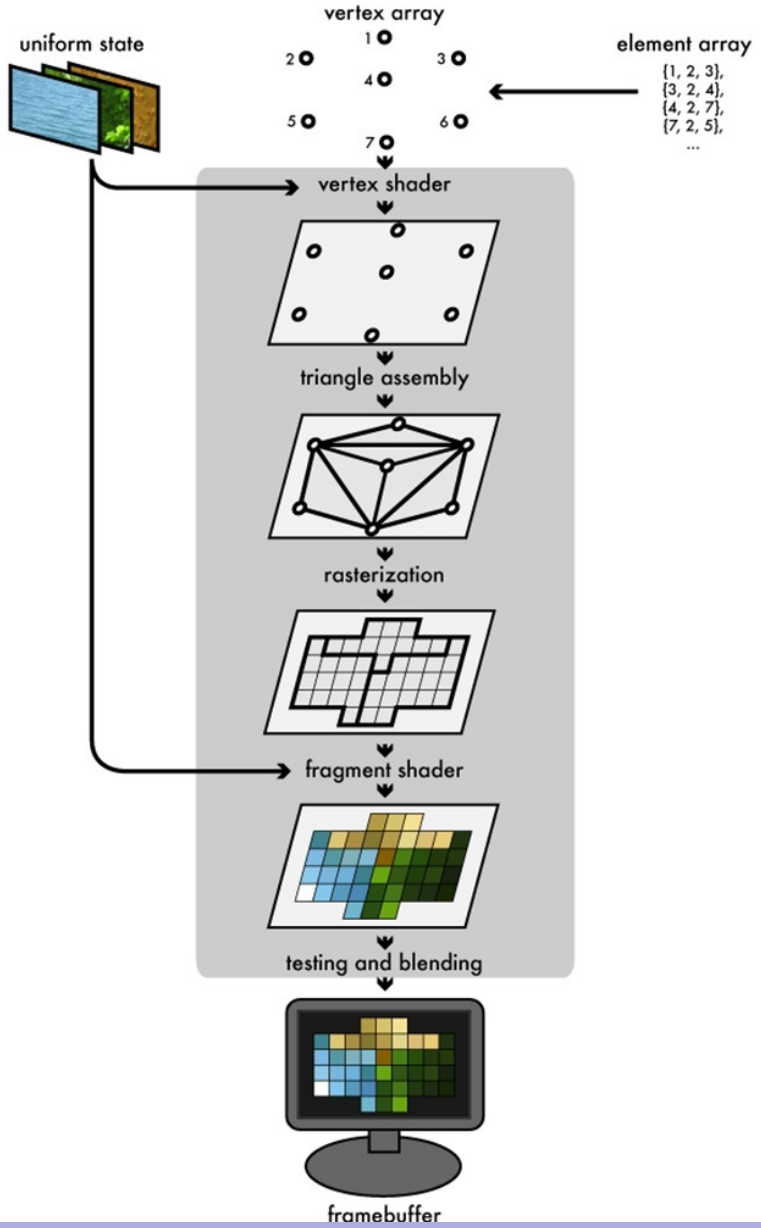
- aus Punkten werden geometrische Primitive erzeugt
  - Punkte, Linien, Dreiecke
- „Rechte Daumen Regel”:



OpenGL Primitive



OpenGL Primitive



Shader Programm

- Vertex-Shader
- Fragment-Shader

Kommunikation zwischen Host-Code und Shadern:

- Uniform-Variablen: Konstant von Programm gesetzt (State-Machine)
- Attribute: Pro Vertex (Punkt / Farbe / Normal etc.)
- Varying: Nur zur Kommunikation zwischen Shader-Stages.

GLWidget:

- `paintGL()` wird periodisch aufgerufen
  - Zeichnen der Szene findet hier statt

- `initializeGL()` ausgeführt vor dem ersten Aufruf von `paintGL()`

Erweiterung: Erstellen Sie einen Würfel.

- Erweitern Sie das Dreieck zu einem Quadrat
- Erstellen Sie fünf weitere Quadrate in entsprechender Position/Orientierung

Hinweis: Würfel wird konstant von vorne gezeigt, d.h. nur eine Fläche sichtbar.

- Fügen Sie eine Drehung um die X und Z Achsen hinzu.
- Der entsprechende Code wird auf der nächsten Folie gezeigt.

Header in GLWidget.cpp einfügen

```
#include <glm/gtc/matrix_transform.hpp>
```

paintGL():

```
[...]
// wichtig:   nach(!) glUseProgram() aber vor(!) glDrawArrays()
static float angle = 0.0;
angle += 0.1;
glUniform1f (glGetUniformLocation (prg, "angle"), angle*0.25);
glm::mat4 rotX;
rotX = glm::rotate (rotX, angle, glm::vec3(1,0,0));
GLint rotXloc = glGetUniformLocation (prg, "rotX");
glUniformMatrix4fv (rotXloc, 1, GL_FALSE, &rotX[0][0]);
[...]
```

vs.glsl:

```
[...]
uniform float angle;
uniform mat4 rotX;
[...]
void main(void) {
   mat4 rotZ = mat4(cos(angle), -sin(angle),0, 0,
                    sin(angle), cos(angle), 0, 0,
                    0, 0, 1, 0,
                    0 ,0 ,0, 1);
   gl_Position = rotX * rotZ * pos;
   [...]
}
```

Links / Dokumentation

Qt:

- http://doc.qt.io

OpenGL:

- https://www.opengl.org/sdk/docs/man/docbook4
- https://www.opengl.org/sdk/docs/

GLSL:

- https://www.opengl.org/documentation/glsl/

GLM:

- http://glm.g-truc.net