

# OpenGL

Sebastian Mendez

Nicht vor 5 Minuten fertig gemacht

# Ein Bisschen Geschichte

## Was ist OpenGL?

- ▶ 3D Rendering API
- ▶ Plattform-, Hardware- und Programmiersprachen-unabhängige Schnittstelle
- ▶ OpenGL implementiert die Grafik-Pipeline
  - ▶ geometrische Primitive: Punkte, Linien, Dreiecke, ...
  - ▶ Bild-Primitive: Images und Bitmaps
  - ▶ Texturen, Texturfilterung (Mip-Mapping etc.), z-Buffer
  - ▶ Stencil-Buffer, Accumulation-Buffer, Alpha-Blending
  - ▶ **klassisches OpenGL:** konfigurierbare Pipeline
    - ▶ Blinn-Phong-Beleuchtungsmodell und Gouraud-Shading
  - ▶ **modernes OpenGL:** programmierbare Stufen in der Pipeline
    - ▶ frei programmierbare Geometrieverarbeitung, Schattierungsberechnung und Tessellierung, ...
    - ▶ **Shaders**

# API zum Grafik-Hardware

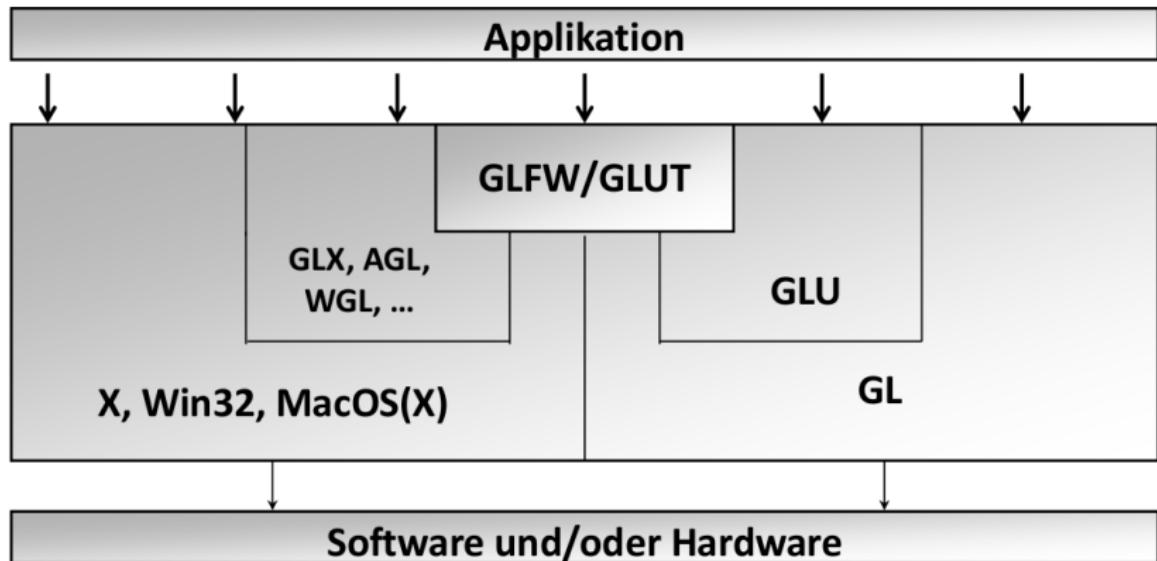


Figure 1: API zum Grafik-Hardware

# Was ist mit der GPU?

- ▶ GPU = Graphic Processing Unit
- ▶ Anders als die CPU
- ▶ NVIDIA Kepler: **2496** Prozessoren
- ▶ SIMD-Modell: Single Instruction, Multiple Threads



Figure 2: GPU Diagram

# An die Arbeit!

- ▶ Erst: OpenGL Kontext initialisieren

# An die Arbeit!

- ▶ Erst: OpenGL Kontext initialisieren
- ▶ Kontext: Sammlung von Zuständen, Framebuffers, ...

# An die Arbeit!

- ▶ Erst: OpenGL Kontext initialisieren
- ▶ Kontext: Sammlung von Zuständen, Framebuffers, ...
- ▶ Wir haben schon unser Window-Handle

In Windows eher....kompliziert...

- ▶ GLEW: GL Extension Wrangler
- ▶ GLUT: GL UTilities
- ▶ Pixelformat feststellen

```
1 PIXELFORMATDESCRIPTOR pfd =
2 {
3     sizeof(PIXELFORMATDESCRIPTOR),
4     1,
5     PFD_DRAW_TO_WINDOW|PFD_SUPPORT_OPENGL|PFD_DOUBLEBUFFER,
6     PFD_TYPE_RGBA,
7     32,
8     0,0,0,0,0,0,0,0,0,0,0,0, // useless parameters
9     16,
10    0,0,0,0,0,0,0
11 };
```

- ▶ Context-attribute

- ▶ Context-attribute
- ▶ Pointer zu den Funktionen holen...

- ▶ Context-attribute
- ▶ Pointer zu den Funktionen holen...
- ▶ glewInit()

- ▶ Context-attribute
- ▶ Pointer zu den Funktionen holen...
- ▶ glewInit()
- ▶ ACHTUNG MIT OOP!!!

- ▶ Context-attribute
- ▶ Pointer zu den Funktionen holen...
- ▶ glewInit()
- ▶ ACHTUNG MIT OOP!!!
- ▶ Endlich können wir malen

## Allgemeine Schritte:

- ▶ Szene vorbereiten

## Allgemeine Schritte:

- ▶ Szene vorbereiten
- ▶ Data erstellen -> zum GPU schicken!

## Allgemeine Schritte:

- ▶ Szene vorbereiten
- ▶ Data erstellen -> zum GPU schicken!
- ▶ Malen (...wiederholen)

## Allgemeine Schritte:

- ▶ Szene vorbereiten
- ▶ Data erstellen -> zum GPU schicken!
- ▶ Malen (...wiederholen)
- ▶ Aufräumen

# Szene vorbereiten

- ▶ Shader(-programme) kompilieren

# Szene vorbereiten

- ▶ Shader(-programme) kompilieren
- ▶ Was ist aber ein Shader(-programm)?

## Die Pipeline

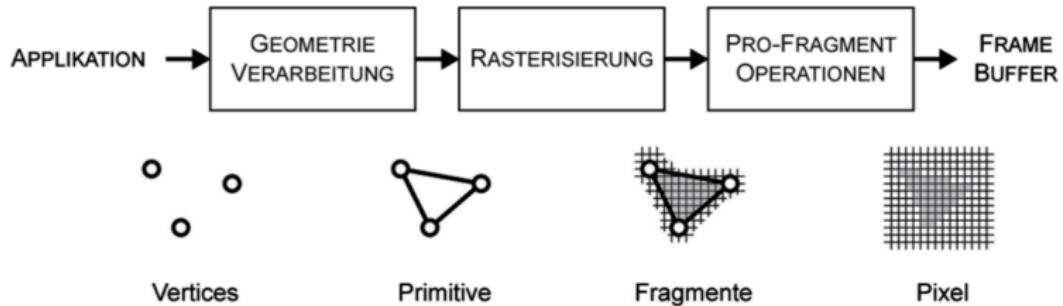


Figure 3: Die Grafik Pipeline

## Die Pipeline

Daten werden Schrittweise verarbeitet. (Fließband)

- ▶ Vertices kommen von der App rein

## Die Pipeline

Daten werden Schrittweise verarbeitet. (Fließband)

- ▶ Vertices kommen von der App rein
- ▶ Vertices werden zu Primitiven zusammengetan

## Die Pipeline

Daten werden Schrittweise verarbeitet. (Fließband)

- ▶ Vertices kommen von der App rein
- ▶ Vertices werden zu Primitiven zusammengetan
- ▶ Primitiven werden zu Fragmenten rasterisiert (2D Projiziert)

## Die Pipeline

Daten werden Schrittweise verarbeitet. (Fließband)

- ▶ Vertices kommen von der App rein
- ▶ Vertices werden zu Primitiven zusammengetan
- ▶ Primitiven werden zu Fragmenten rasterisiert (2D Projiziert)
- ▶ Fragmente geben die endgültige Pixelfarben

## Geometrieverarbeitung: Vertex Shader



Figure 4:

- ▶ Transformation *einzelner* Vertices und deren Attribute
  - ▶ Keine Vertex-Erzeugung, -Lösung, oder Information über andere Vertices

- ▶ Attribute, die für Fragmente interpoliert werden sollen
  - ▶ Für Beleuchtung
  - ▶ Normalen

```
1 void main() {  
2     gl_Position = gl_ModelViewProjectionMatrix *  
                 gl_Vertex;  
3 }
```

# Geometrieverarbeitung: Assembly + Tessellation

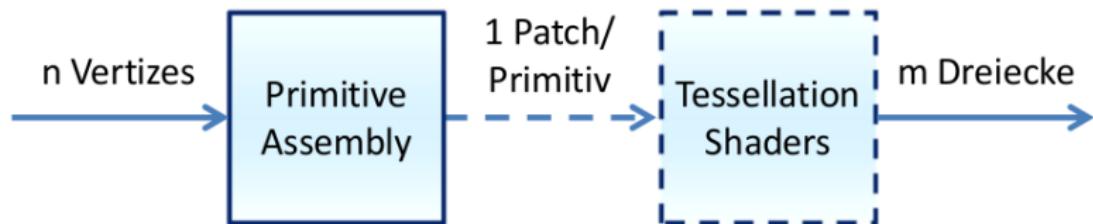


Figure 5:

- ▶ Primitive Assembly ist **nicht programmierbar**. Von der App festgelegt
- ▶ Tessellation (optional): Unterteilung der Primitiven

# Tessellation



Figure 6: Vor der Tessellation

# Tessellation

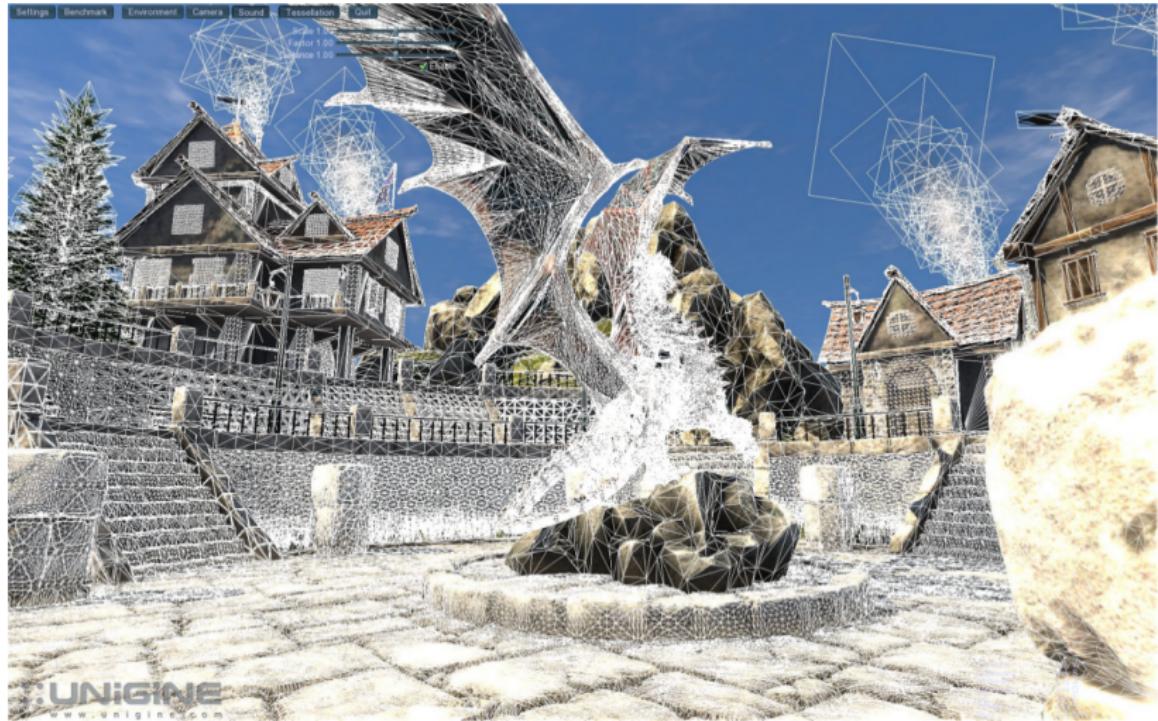


Figure 7: Nach der Tessellation

# Geometrieverarbeitung: Geometry Shader



Figure 8:

- ▶ Geometry Shader (optional):
  - ▶ kann Primitive vervielfachen, entfernen, umwandeln
- ▶ Anschließend: perspektivistische Division & Rasterisierung (siehe später)

# Fragmentverarbeitung

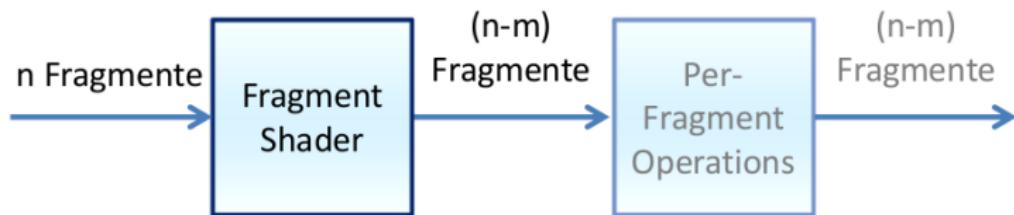


Figure 9:

- ▶ Für jedes Fragment, wird ein Fragment Shader ausgeführt
  - ▶ Berechnung von: Farbe, Transparenz, und Tiefe (optional)
    - ▶ Beleuchtung pro Pixel

```
1 void main() {  
2     gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);  
3 }
```

# Wie mache ich dann ein Shaderprogramm?

- ▶ `glCreateProgram`
- ▶ Pro Shader: `glAttachShader`
  - ▶ `glCreateShader`
  - ▶ `glShaderSource`
  - ▶ `glCompileShader`
- ▶ `glLinkProgram`
- ▶ `glUseProgram` (beim Zeichnen)

Vorarbeit fertig! Jetzt Daten übertragen!

# OpenGL-Objekte

- ▶ Keine OOP-Objekte
- ▶ Kapseln Daten für die Grafikkarte
- ▶ Allgemeines Rezept:

```
1 GLuint handle;  
2 glGen{Textures|Buffers|etc.}(Anzahl, &someHandle);  
    //Erzeuge Objekt  
3 glBind(<target>, someHandle); //Objekt selektieren  
4 //Daten übertragen, falls nötig  
5 glBind(<target>, 0); //Objekt deselektieren WICHTIG!  
6 glDelete(Anzahl, &someHandle); //Objekt löschen
```

## Beispiel

```
1 float* vert = new float[9];
2
3 vert[0] = -0.3; vert[1] = 0.5; vert[2] = -1.0;
4 vert[3] = -0.8; vert[4] = -0.5; vert[5] = -1.0;
5 vert[6] = 0.2; vert[7] = -0.5; vert[8] = -1.0; //Daten im
       CPU!!!
6
7 glGenBuffers(1, vboID);
8
9 glBindBuffer(GL_ARRAY_BUFFER, vboID);
10
11 glBufferData(GL_ARRAY_BUFFER, 9 * sizeof(GLfloat), vert,
               GL_STATIC_DRAW);
12
13 glVertexAttribPointer((GLuint)0, 3, GL_FLOAT, GL_FALSE,
                      0, 0);
14 glEnableVertexAttribArray(0);
15
```



# Malen

- ▶ `glViewport(0, 0, w, h)`: “Bildschirm” einstellen.

# Malen

- ▶ `glViewport(0, 0, w, h)`: “Bildschirm” einstellen.
  - ▶ Wenn sich die Fenstergröße ändert!

# Malen

- ▶ `glViewport(0, 0, w, h)`: "Bildschirm" einstellen.
  - ▶ Wenn sich die Fenstergröße ändert!
- ▶ `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`: Framebuffer löschen

# Malen

- ▶ `glViewport(0, 0, w, h)`: "Bildschirm" einstellen.
  - ▶ Wenn sich die Fenstergröße ändert!
- ▶ `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`: Framebuffer löschen
- ▶ Malen

# Malen

```
1 glBindVertexArray(vaoID);
2 glDrawArrays(GL_TRIANGLES, 0, 3);
3
4 glBindVertexArray(0);
5
6 SwapBuffers();
```

Warum SwapBuffers?