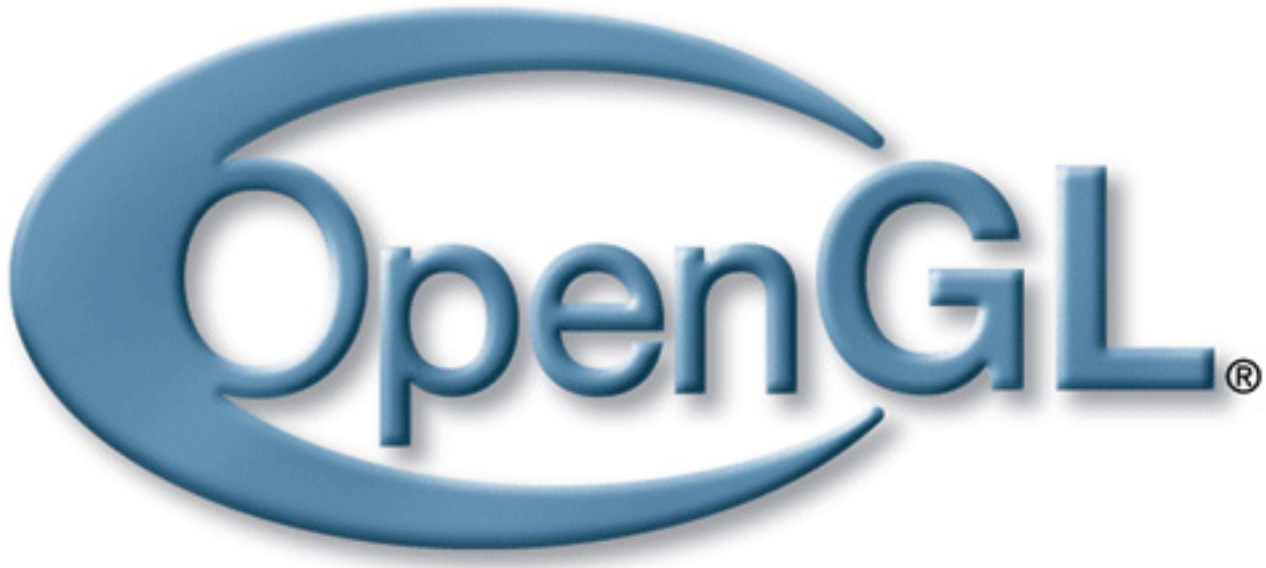
OpenGL - Teil 4

- Shader in OpenGL -



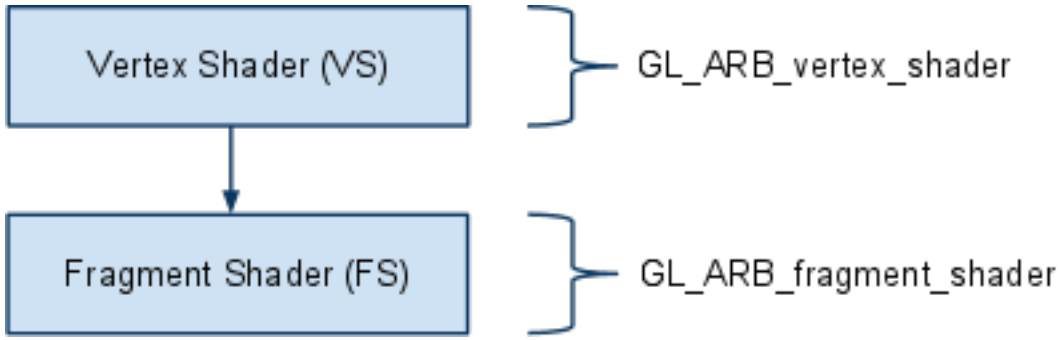
Bernhard Bittorf und Andy Reimann

OpenGL - Was sind Shader?

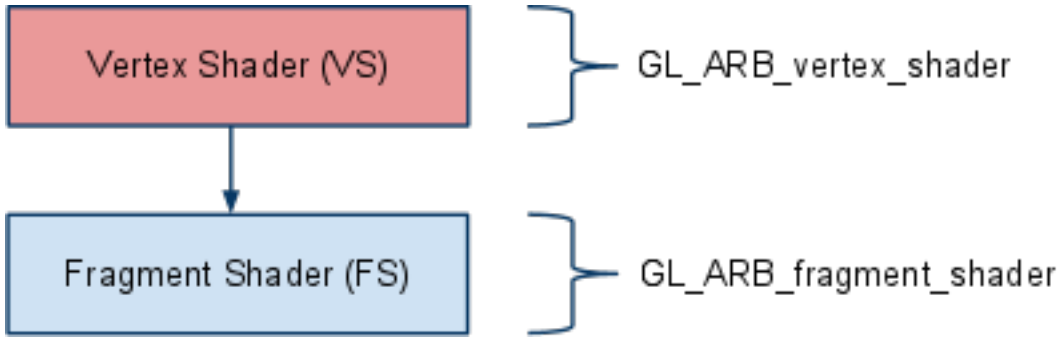
- kleine Programme, die auf der GPU laufen
- können auf verschiedenen Ebenen (Stages) laufen
  - verarbeiten Vertexdaten oder Pixeldaten
  - produzieren Vertexdaten oder Pixeldaten
- machen die *'Fixed-Functionality-Pipeline'* (FFP) zu einer *'Programable Pipeline'* (PP)
- *Wir entscheiden, wie Pixeldaten und Vertexdaten auszusehen haben, nicht mehr OpenGL!*

OpenGL 2.x - Die Pipeline

OpenGL 2.x - Die Pipeline



OpenGL 2.x - Die Pipeline



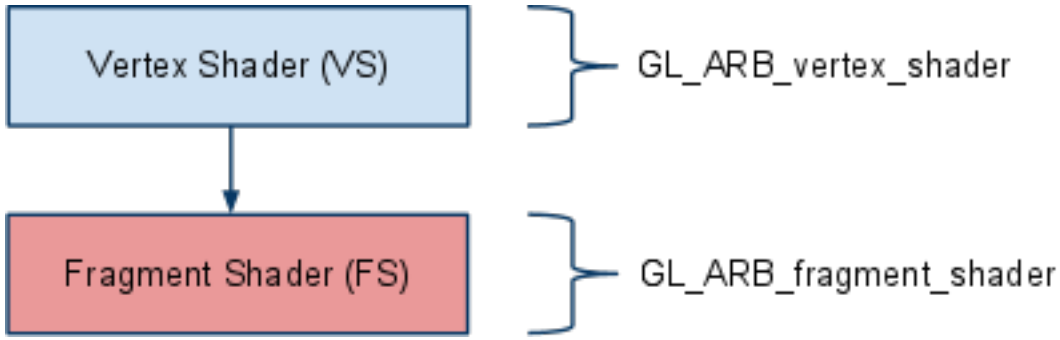
Input

- Farbe, Texkoords, Normale, Vertexposition, Texturen, Matrizen, ...

Output

- finale Vertexposition im Clipspace (Fensterkoordinaten)

OpenGL 2.x - Die Pipeline



Input

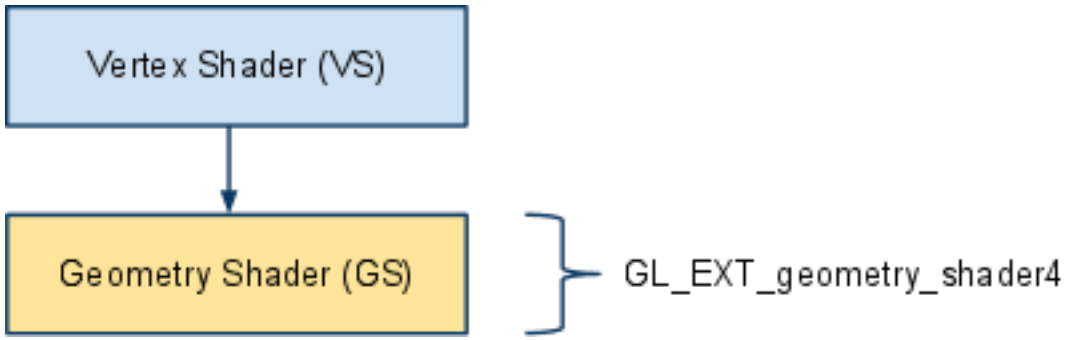
- Farbe, Texkoords, Normale, Vertexposition, Texturen, Matrizen, ...

Output

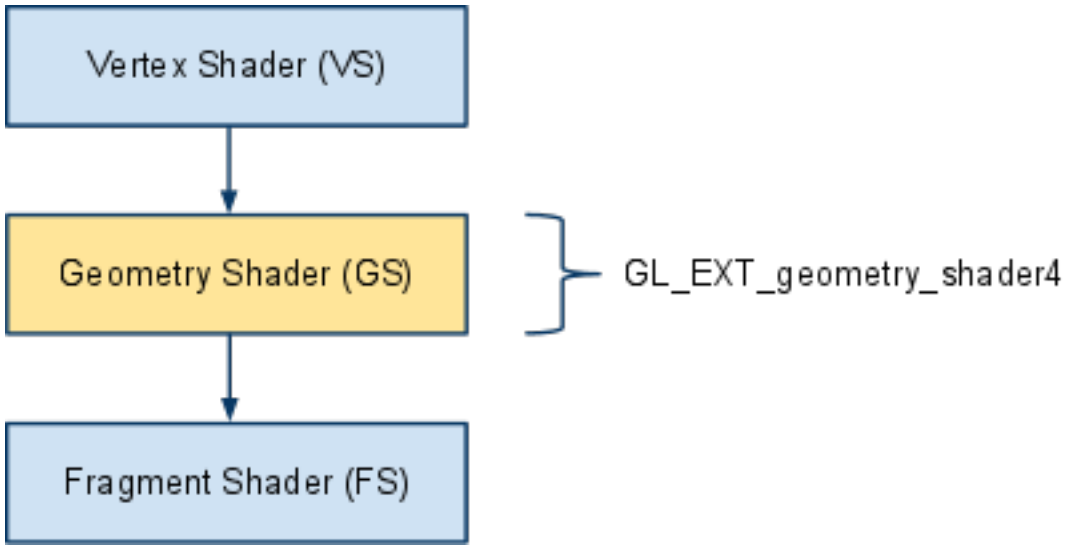
- finale Vertexfarbe

OpenGL 3.x - Die Pipeline

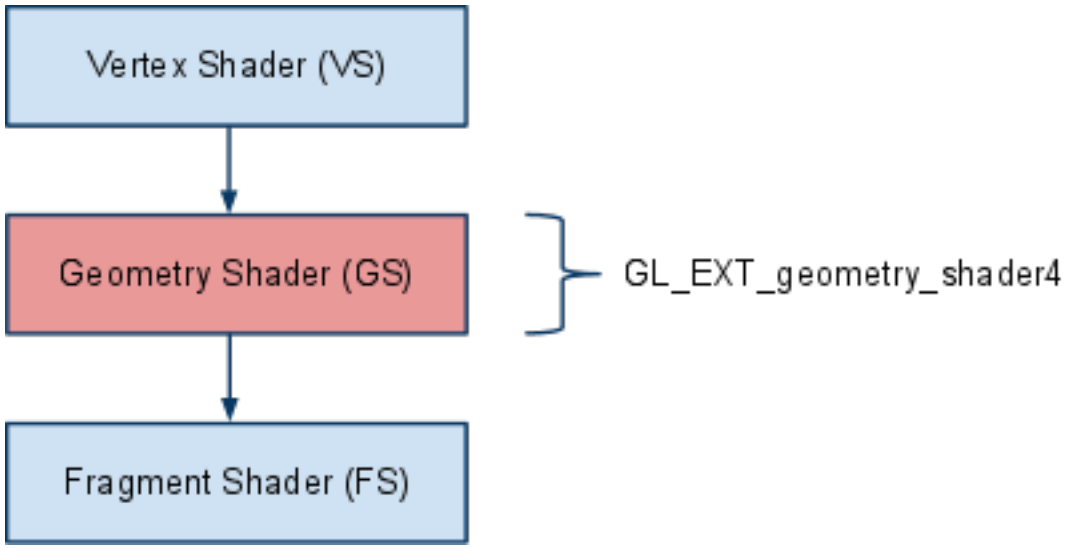
OpenGL 3.x - Die Pipeline



OpenGL 3.x - Die Pipeline



OpenGL 3.x - Die Pipeline



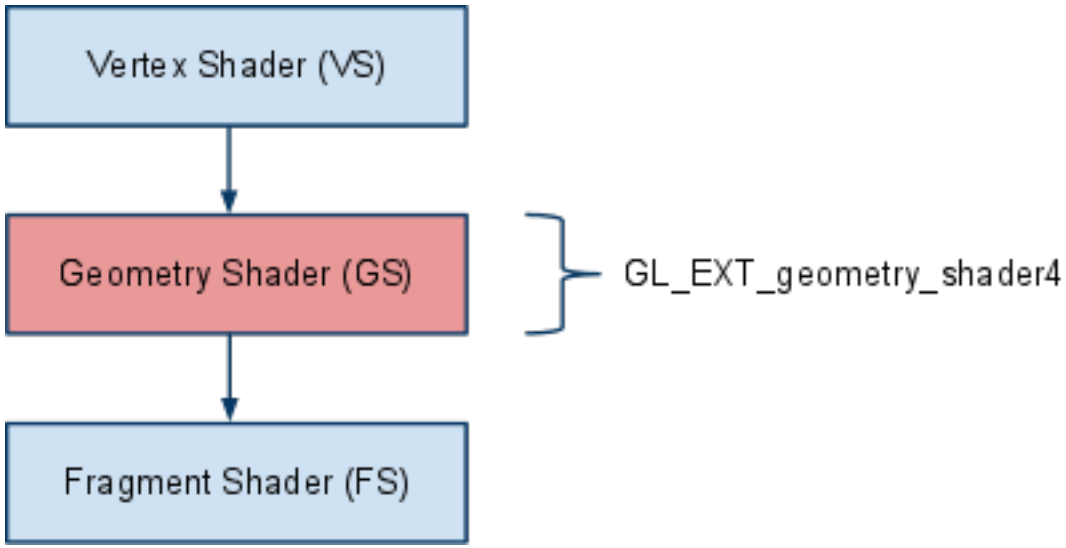
Input

- Positionen, Farben, TexCoords eines Primitivs (Triangle, Line, Point) aus Vertex Shader Stage

Output

- Duplizierte Primitive mit selbst bestimmten Positionen , Farben, TexCoords

OpenGL 3.x - Die Pipeline

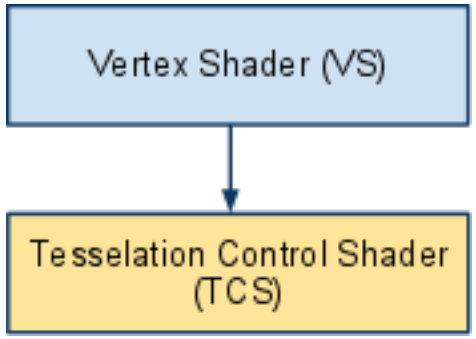


Limitationen

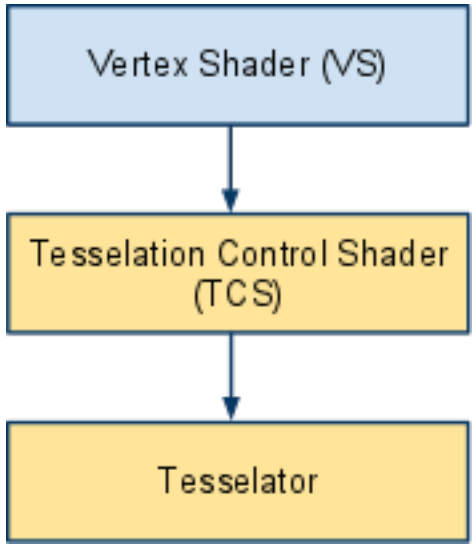
- Vordefinierte Anzahl an zu erzeugenden Primitiven
- Speicherverbrauch auch wenn Primitive nicht erzeugt werden
- nur für kleine Nachbesserungen an Vertexdaten, nicht für High-Change-Algorithmen!

OpenGL 4.x - Die Pipeline

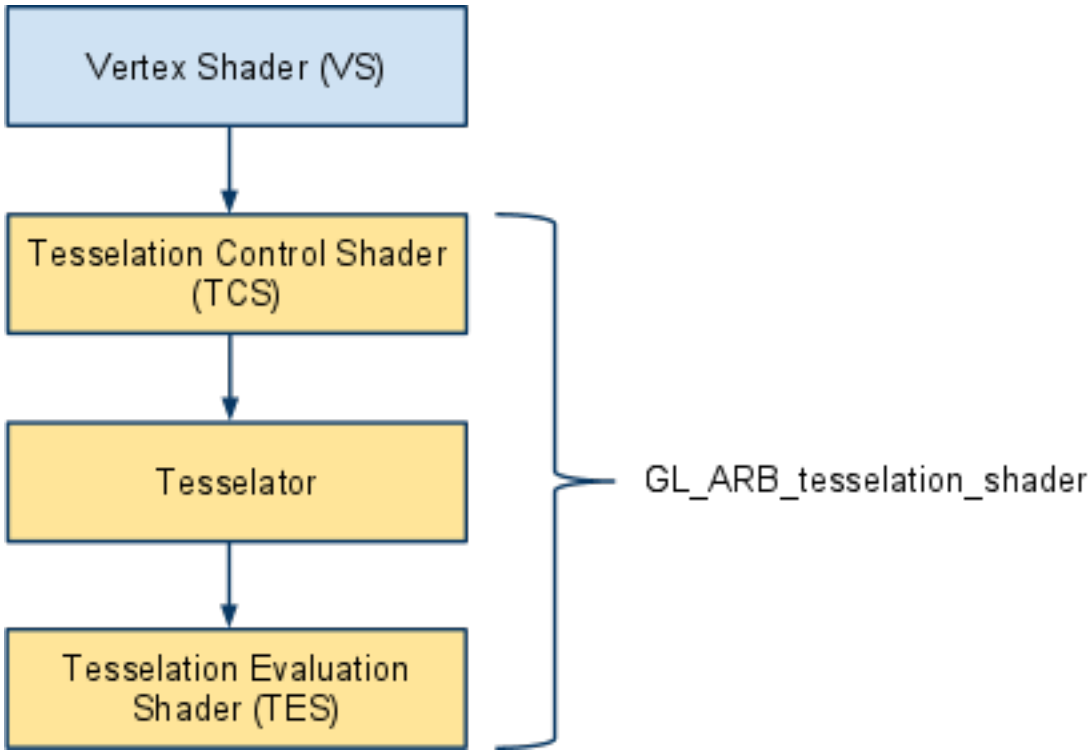
OpenGL 4.x - Die Pipeline



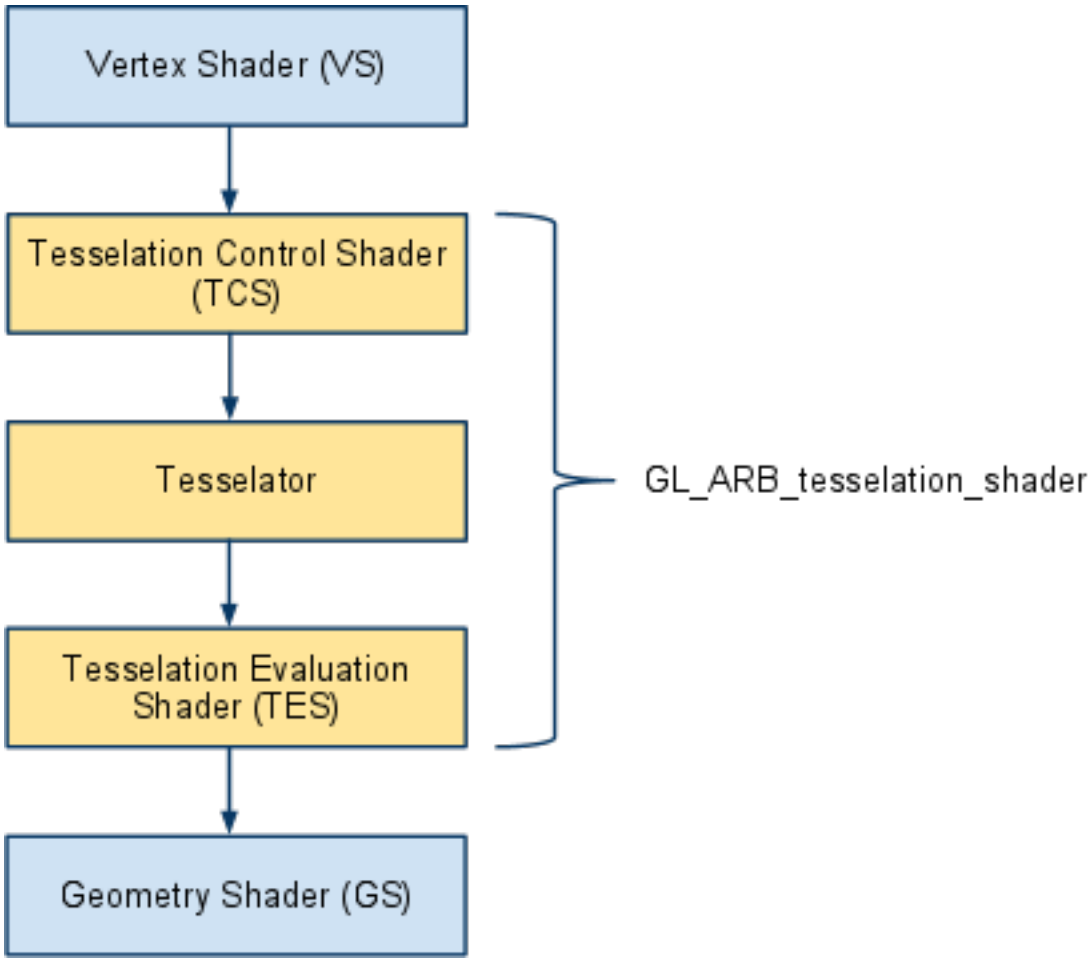
OpenGL 4.x - Die Pipeline



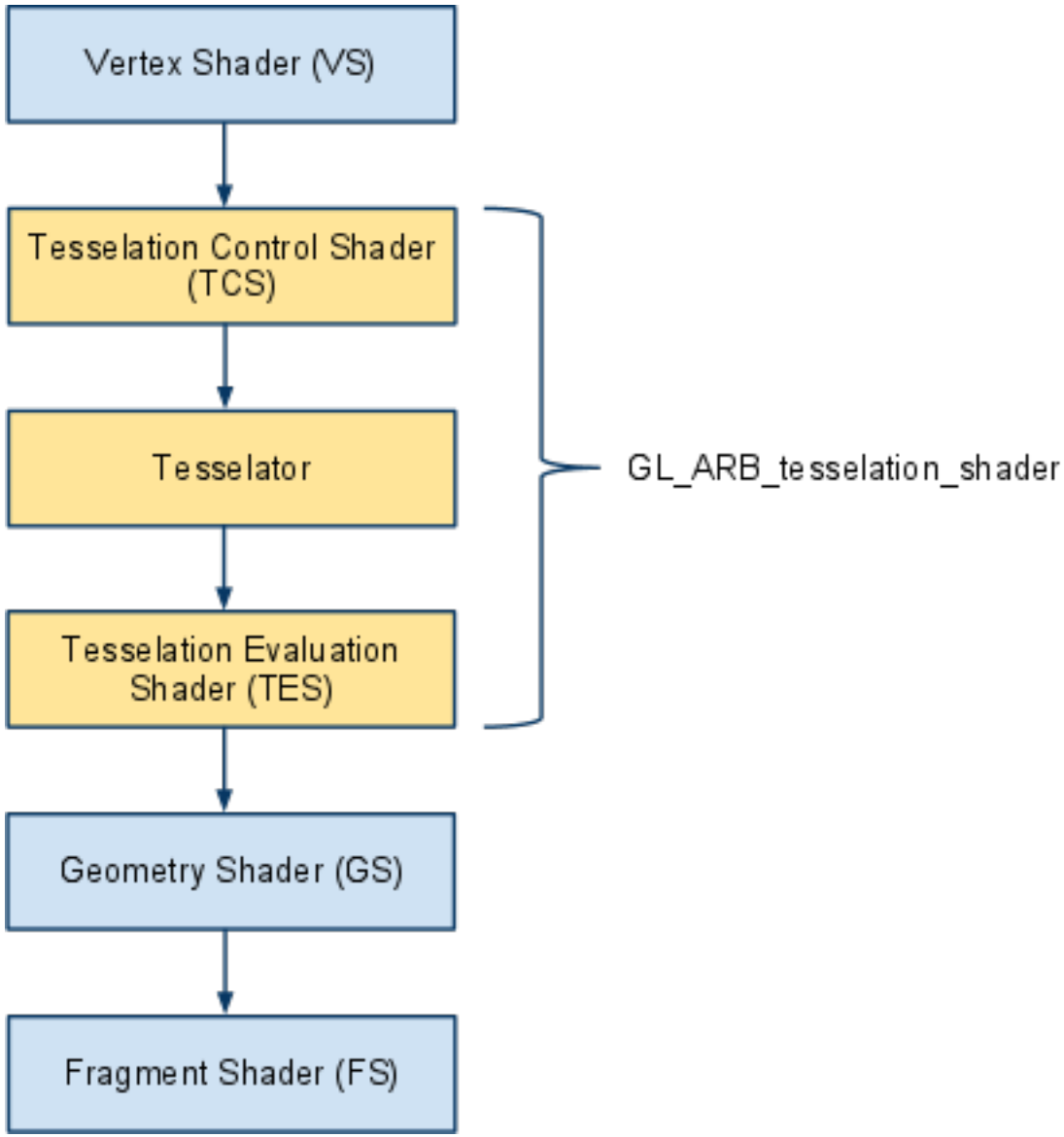
OpenGL 4.x - Die Pipeline



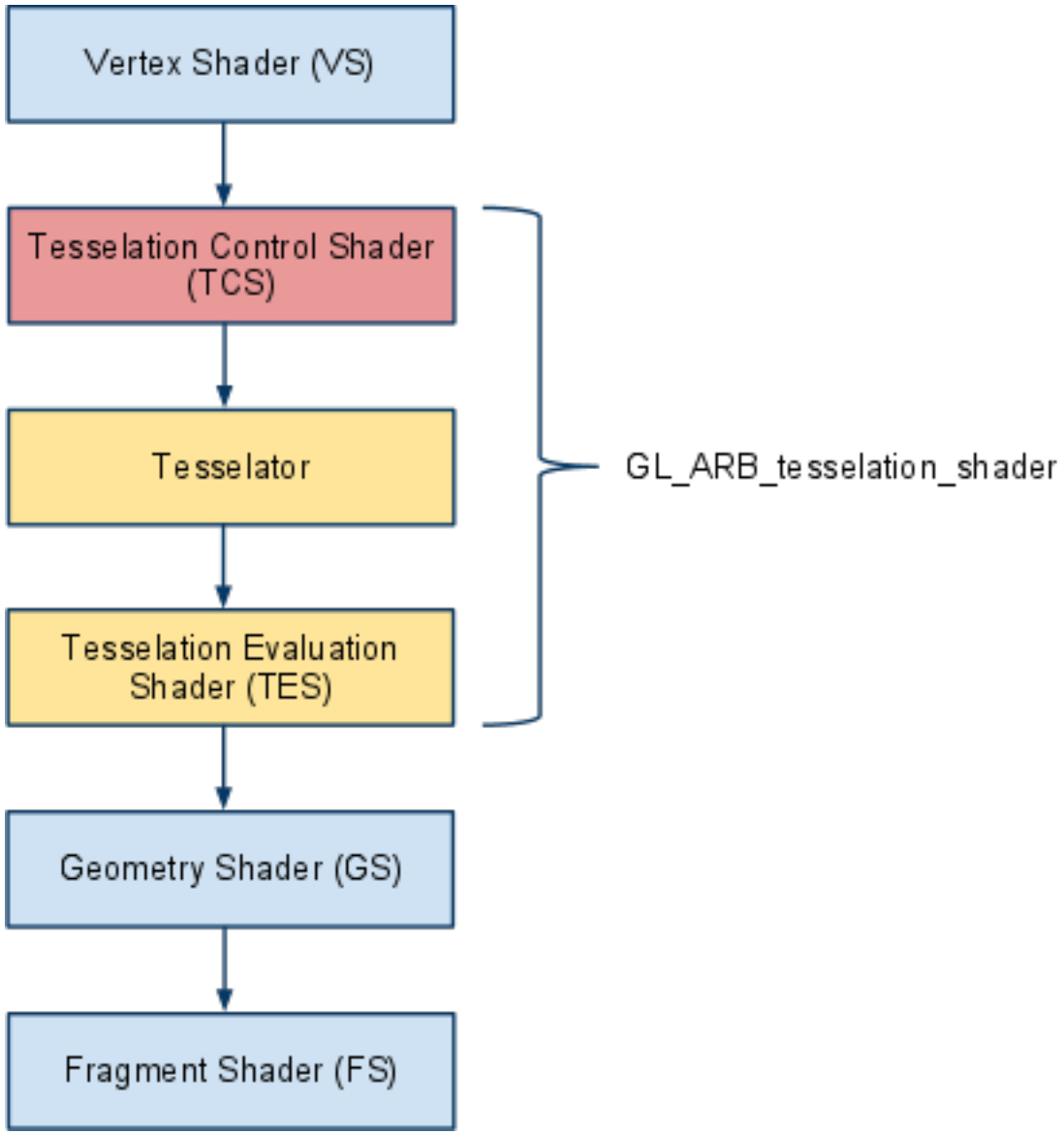
OpenGL 4.x - Die Pipeline



OpenGL 4.x - Die Pipeline



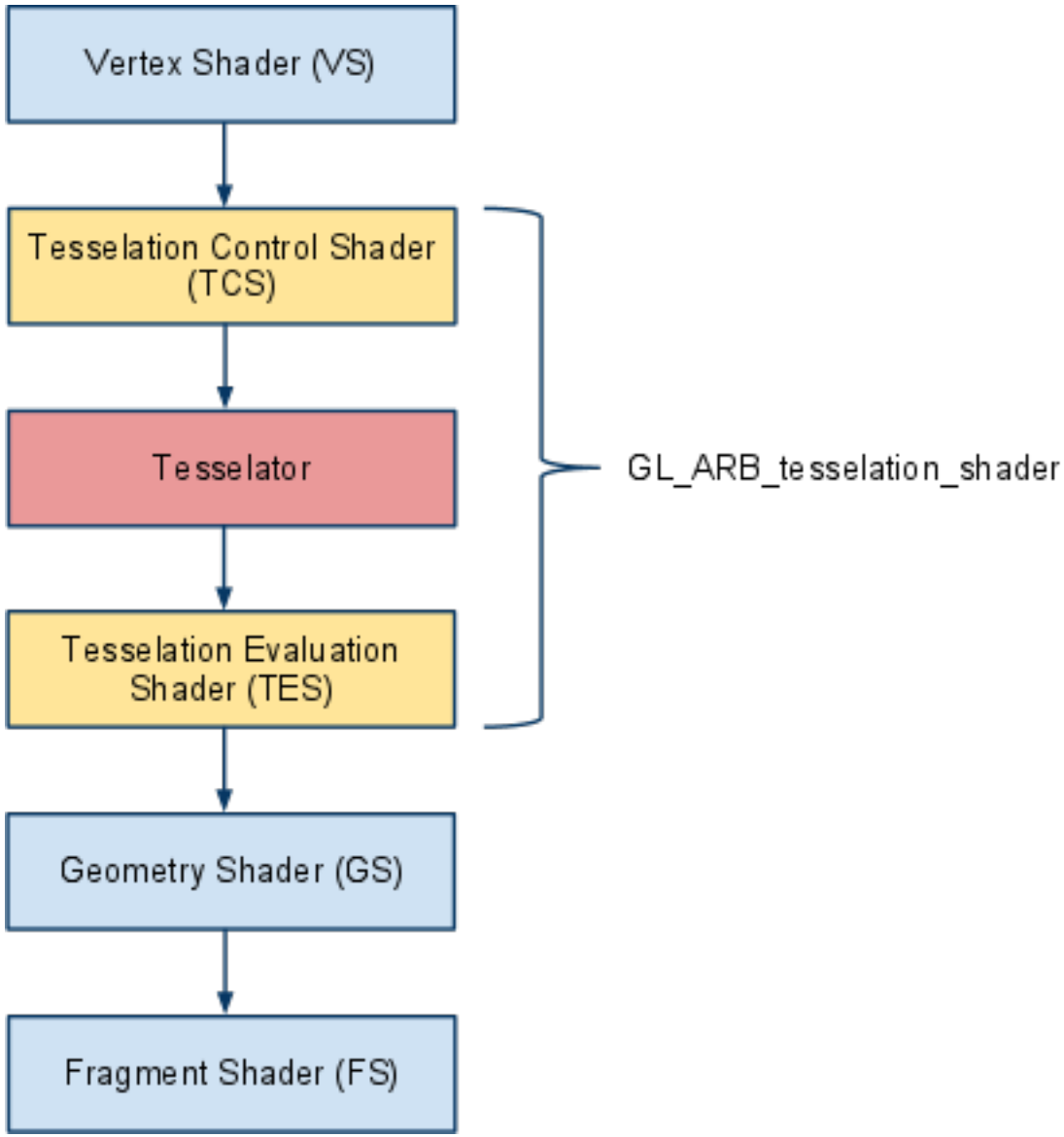
OpenGL 4.x - Die Pipeline



Informationen

- wird für jedes Vertex durchlaufen
- Einziger (neuer) Primitivdrawmode ist GL_PATCH
- berechnet LOD pro Patch

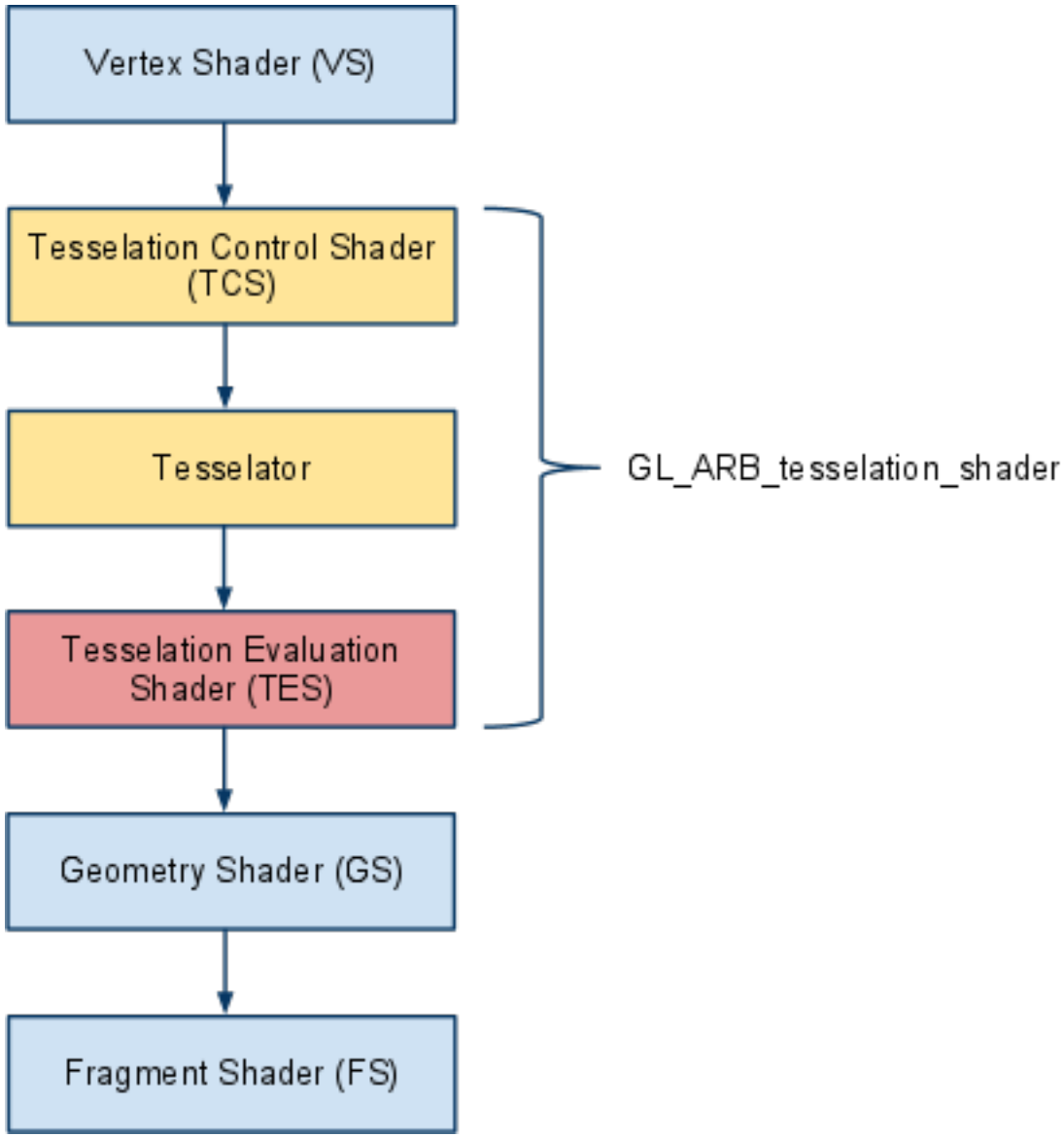
OpenGL 4.x - Die Pipeline



Informationen

- neue fixed function stage
- der eigentliche Tesselierer
- verwendet Tesselierungslevel, um einen Patch in eine Menge von neuen Primitiven zu zerlegen
- Jedem Vertex wird eine [u, v] oder eine [u, v, w] Koordinate zugewiesen

OpenGL 4.x - Die Pipeline



Informationen

- berechnet Position für jedes Vertex, erzeugt in dem Tesselation Control Shader
- Kontrolliert das Tesselierungs-Pattern
- kann Orientierung (cw oder ccw) entscheiden
- kann Primitivmode entscheiden (Points, Lines, Triangles)

OpenGL - GLSL - Was ist das denn?

- "The OpenGL Shading Language"
- erste Herstellerunabhängige Shader Sprache
- eingeführt in OpenGL 2.0 (September 2004)
- an C angelehnt
- erweitert um viele build-in Variablen und Datentypen wie vec2, vec3, vec4, mat4, sampler, ...
- auf GPU Berechnungen hochoptimiert

OpenGL - GLSL - Ein erstes Beispiel

Einfachster Vertexshader:

```
// vordefinierter Einsprungspunkt ist immer main()
void main() {
  // Transformieren des einkommenden gl_Vertex
  gl_Position = gl_Vertex * gl_ModelViewProjectionMatrix;
  // gl_Position ist die von GLSL-Vertexshadern vordefinierte
  // Variable, die immer beschrieben werden muss!
}
```

OpenGL - GLSL - Ein erstes Beispiel

Einfachster Vertexshader:

```
// vordefinierter Einsprungspunkt ist immer main()
void main() {
  // Transformieren des einkommenden gl_Vertex
  gl_Position = gl_Vertex * gl_ModelViewProjectionMatrix;
  // gl_Position ist die von GLSL-Vertexshadern vordefinierte
  // Variable, die immer beschrieben werden muss!
}
```

Einfachster Fragmentshader:

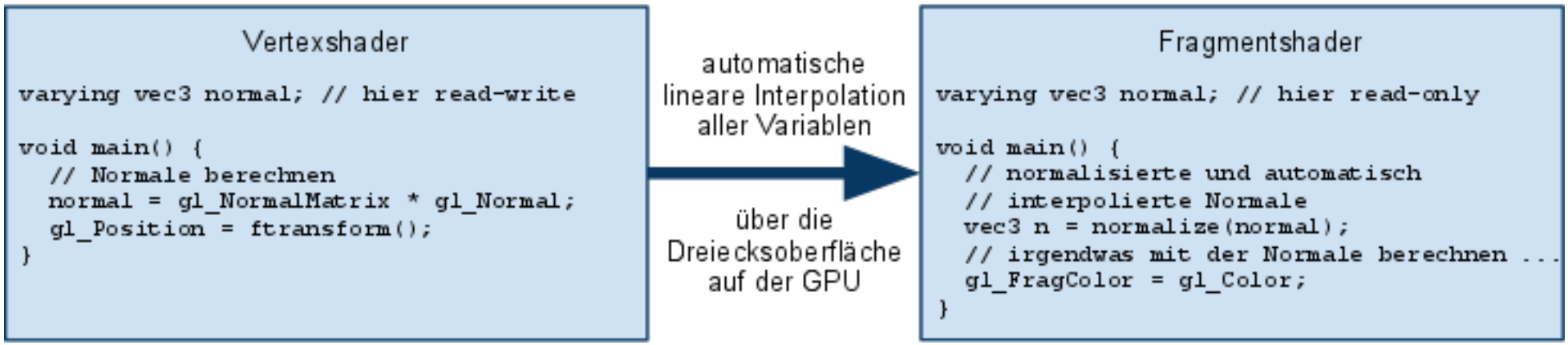
```
// vordefinierter Einsprungspunkt ist immer main()
void main() {
  // Transformieren des einkommenden gl_Vertex
  gl_FragColor = gl_Color;
  // gl_FragColor ist die von GLSL-Fragmentshadern
  // vordefinierte Variable, die immer beschrieben werden
  // muss!
}
```

OpenGL - GLSL - uniforms, attributes and varyings

```
uniform float my_uniform_var;
attribute vec3 my_attribute_var;
varying mat4 my_varying_var;
void main() {
  // ...
}
```

- uniforms und attributes können vom C++ Program gesetzt werden
- alle sind in allen Shader-Stages aus verwendbar
- uniforms sind sogenannte *"Per-Shader Variablen"*
- attributes sind sogenannte *"Per-Vertex Variablen"*
- varyings sind Shader interne Variablen (werden im Vertexshader gesetzt und im Fragmentshader verwendet)

OpenGL - GLSL - varyings



OpenGL - GLSL - varyings

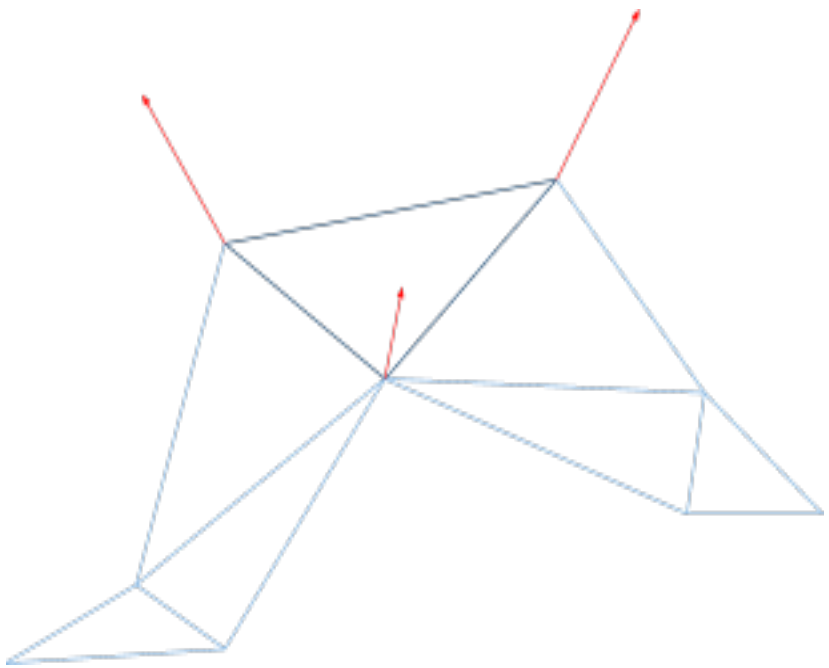
Vertexshader

```
varying vec3 normal; // hier read-write

void main() {
  // Normale berechnen
  normal = gl_NormalMatrix * gl_Normal;
  gl_Position = ftransform();
}
```

automatische lineare Interpolation aller Variablen über die Dreiecksoberfläche auf der GPU

Fragmentshader

```
varying vec3 normal; // hier read-only

void main() {
  // normalisierte und automatisch
  // interpolierte Normale
  vec3 n = normalize(normal);
  // irgendwas mit der Normale berechnen ...
  gl_FragColor = gl_Color;
}
```



OpenGL - GLSL - varyings

Vertexshader

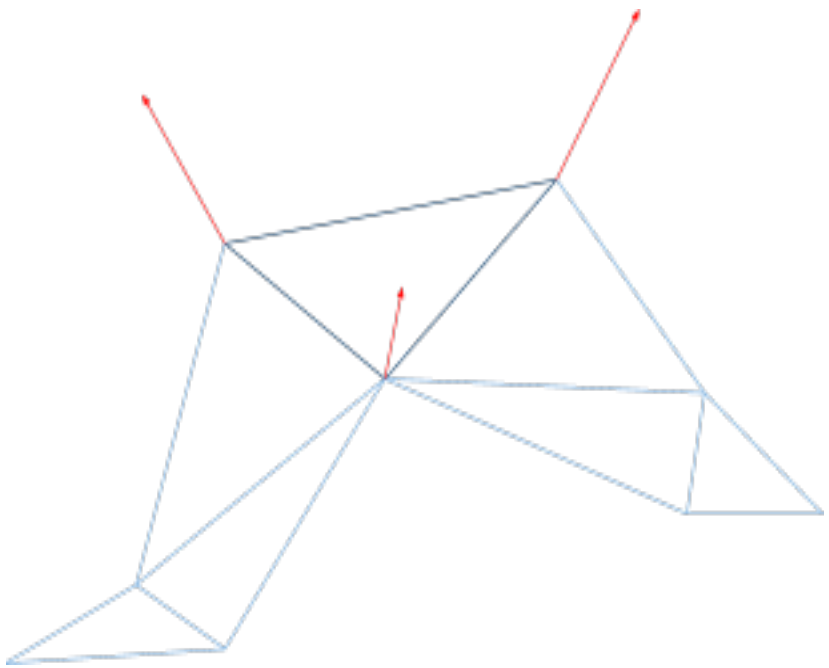
```
varying vec3 normal; // hier read-write

void main() {
  // Normale berechnen
  normal = gl_NormalMatrix * gl_Normal;
  gl_Position = ftransform();
}
```

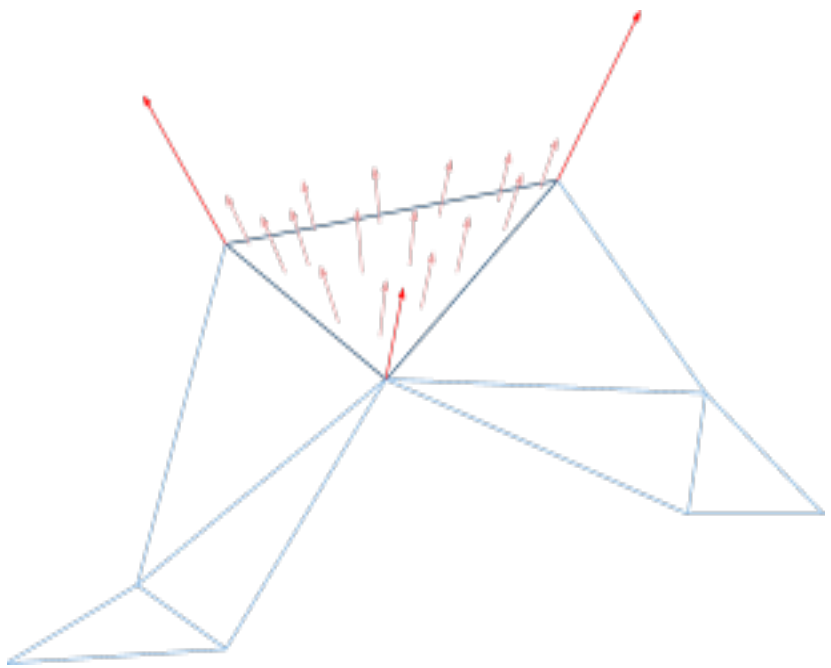
automatische lineare Interpolation aller Variablen

über die Dreiecksoberfläche auf der GPU

Fragmentshader

```
varying vec3 normal; // hier read-only

void main() {
  // normalisierte und automatisch
  // interpolierte Normale
  vec3 n = normalize(normal);
  // irgendwas mit der Normale berechnen ...
  gl_FragColor = gl_Color;
}
```



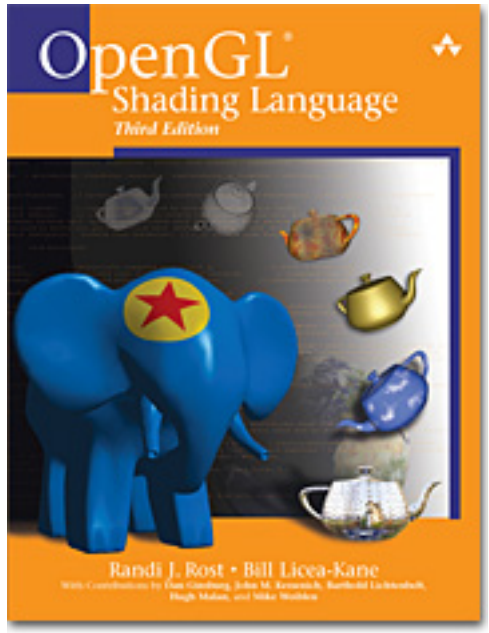


OpenGL - Einen Shader laden

Lighthouse 3D bietet ein vollständiges Tutorial zum Laden und Testen eines Shaders mit Kompilierfehleranzeige (auf GPU Ebene nicht ganz einfach): Lighthouse 3D GLSL Shader Tutorial

OpenGL - GLSL Referenzen

- *The Orange Book* (alle schnell in die Bibo ;))



- *GLSL v1.50 rev.9 Spec* und *OpenGL 3.2 und GLSL 1.50 Quick reference Guide*
- *GLSL v1.50 rev.9 Spec* und *OpenGL 3.2 und GLSL 1.50 Quick reference Guide*
- *GLSL v4.00 rev.8 Spec* und *OpenGL 4.0 und GLSL 4.00 Quick reference Guide*
- Das neue *OpenGL 4.1 und GLSL 4.10 Quick reference Guide*

OpenGL - Die neue Aufgabe

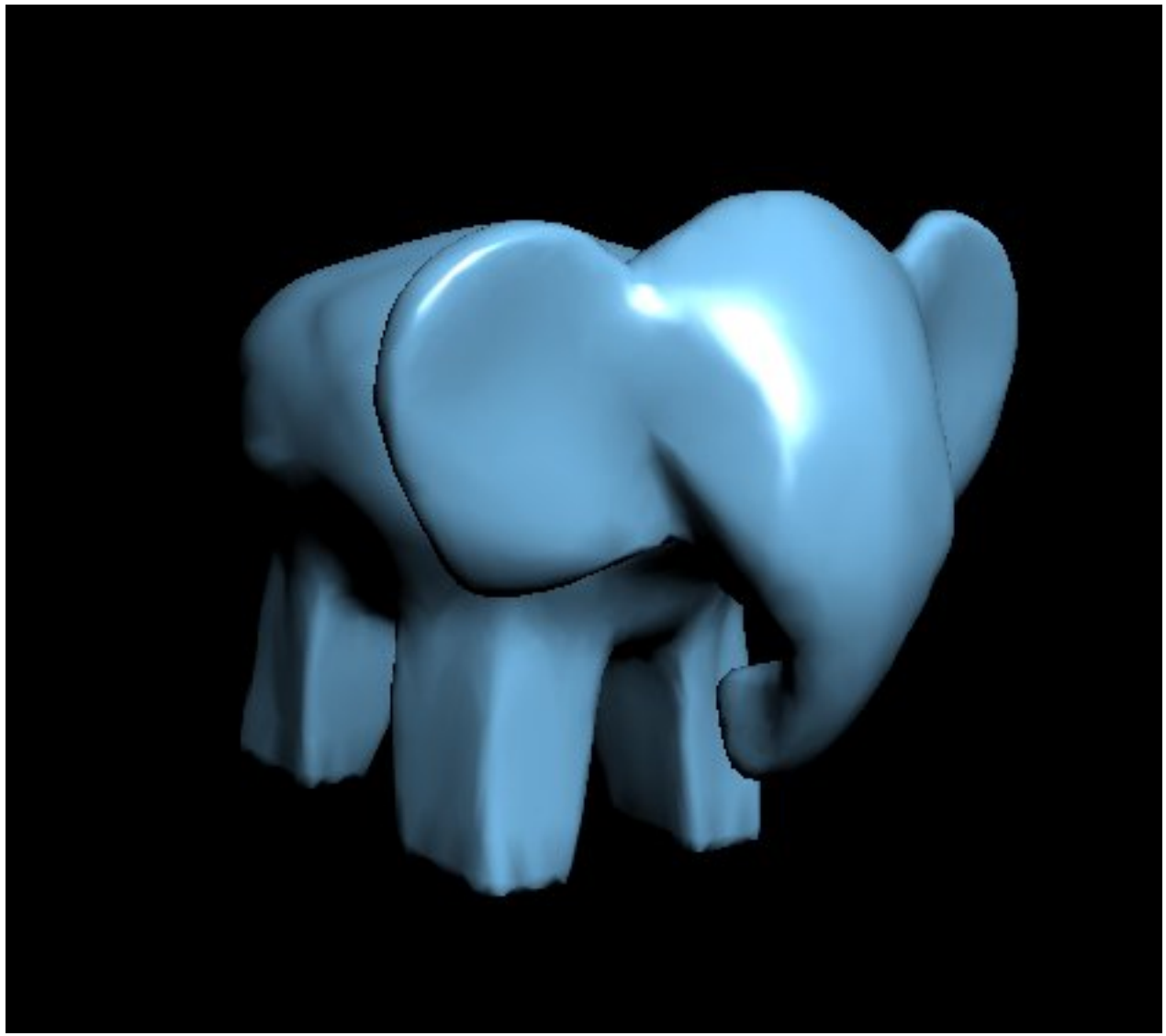
Implementieren Sie mindestens 3 Shader aus der folgenden Liste:

- Farbshader (z.B. Sepia)
- Toonshading
- Blinn-Phong
- Minnaert-Shading
- Partikel
- Shadow Mapping
- Bump Mapping
- Blur oder Radial Blur
- Fog
- Pulsating Objects / Flags / Waves
- HDR/Bloom Shader

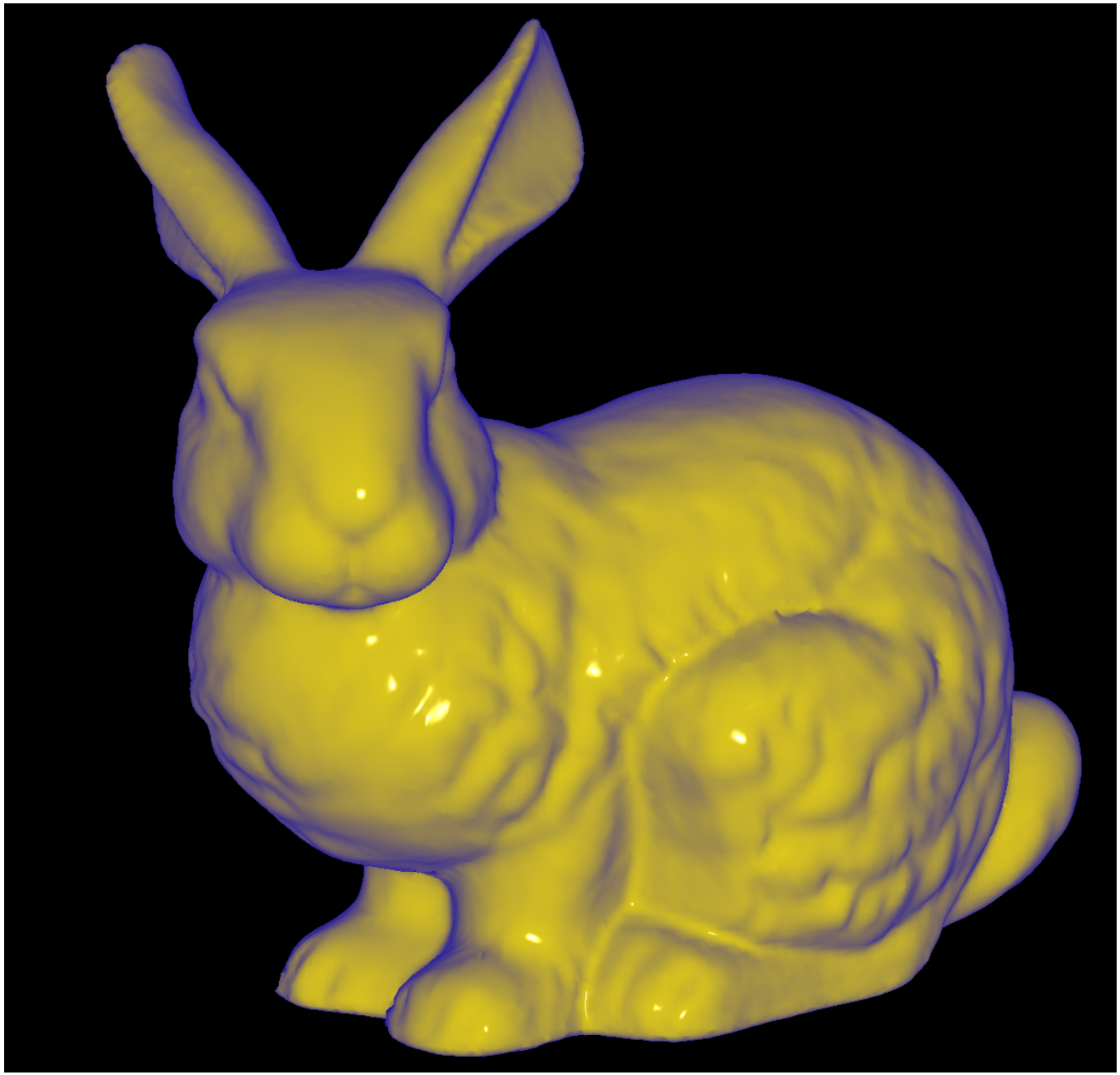
OpenGL - Sepia Shader

OpenGL - Toon Shader

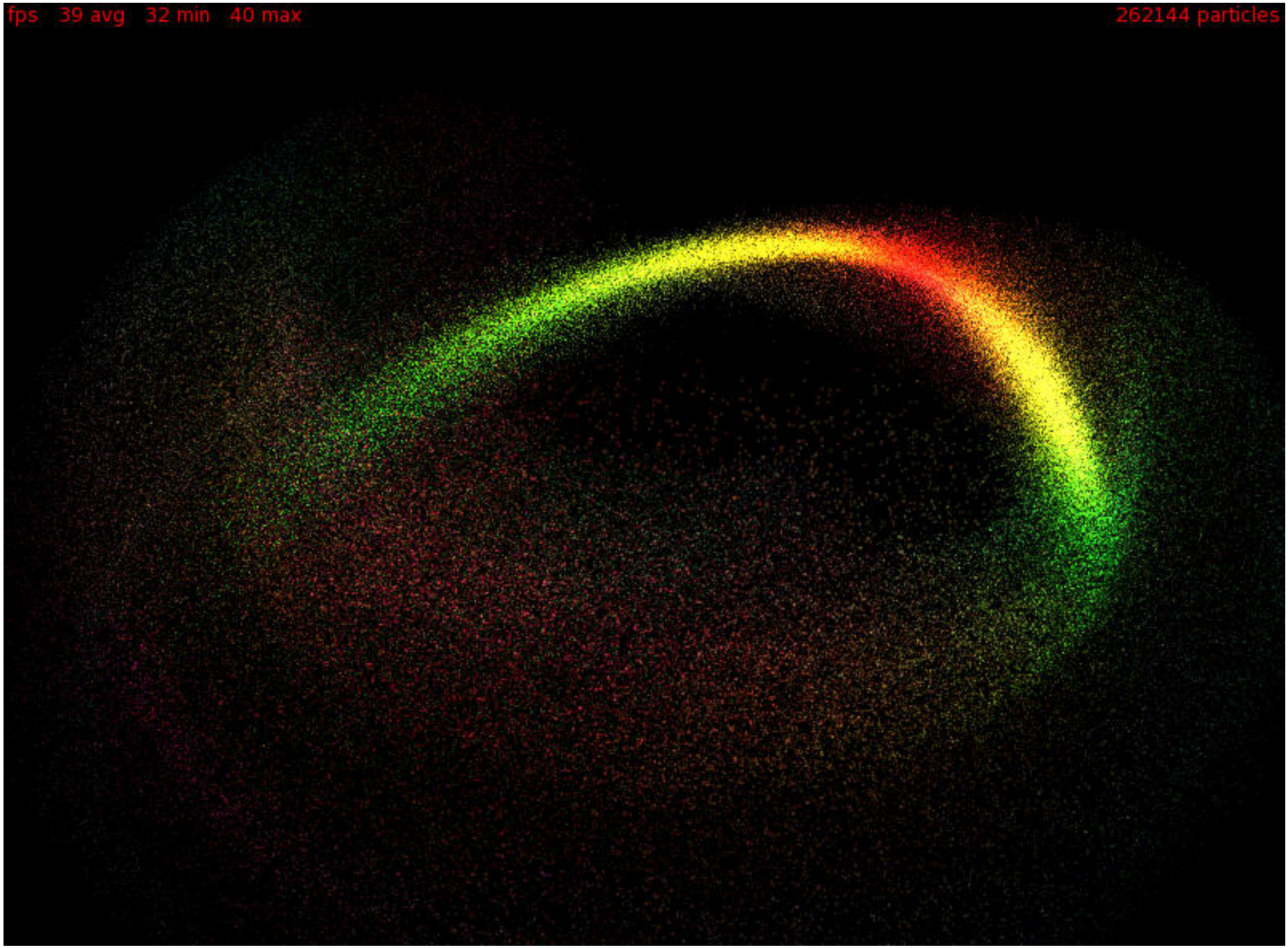
OpenGL - Blinn-Phong Shader



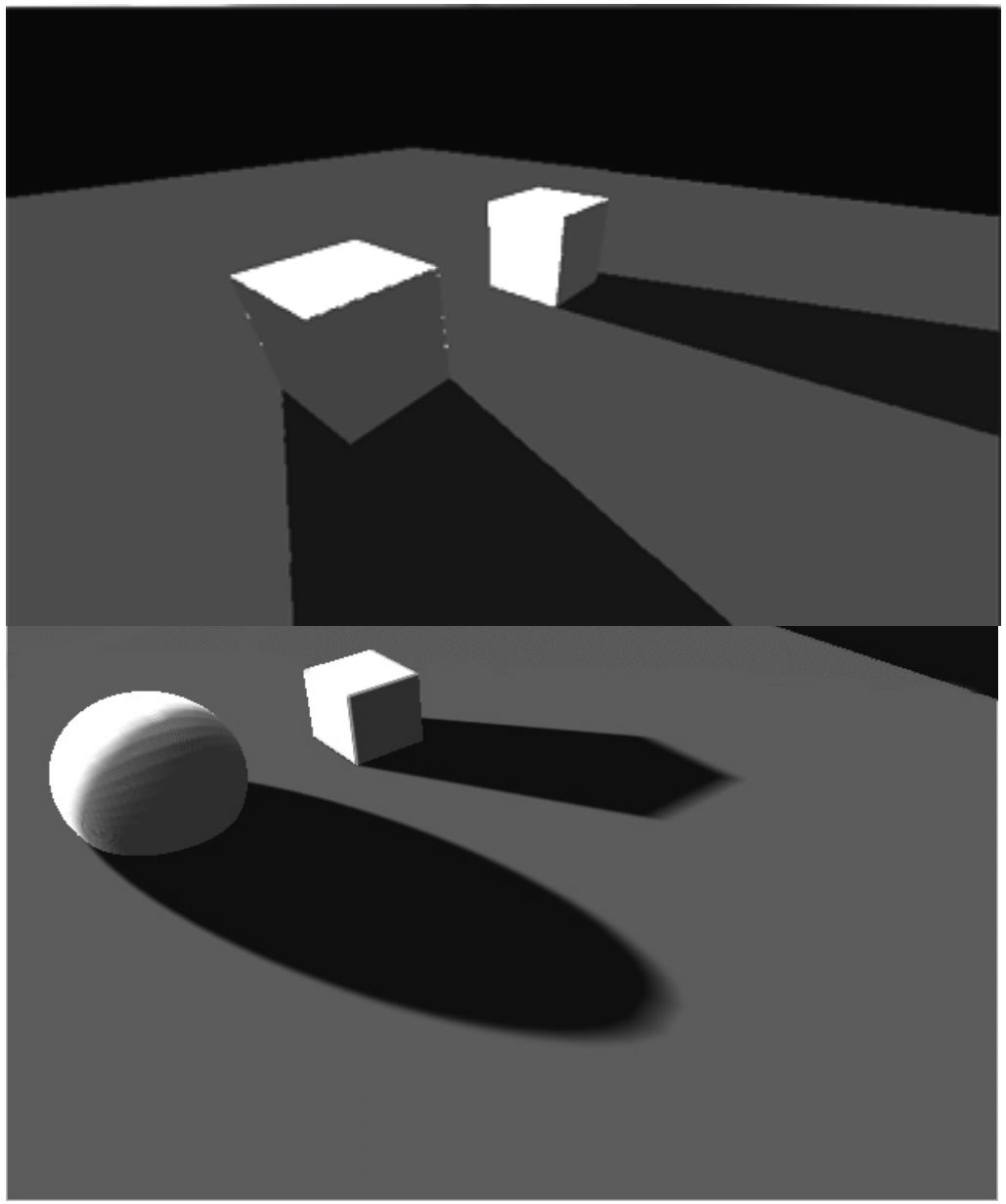
OpenGL - Minnaert Shader



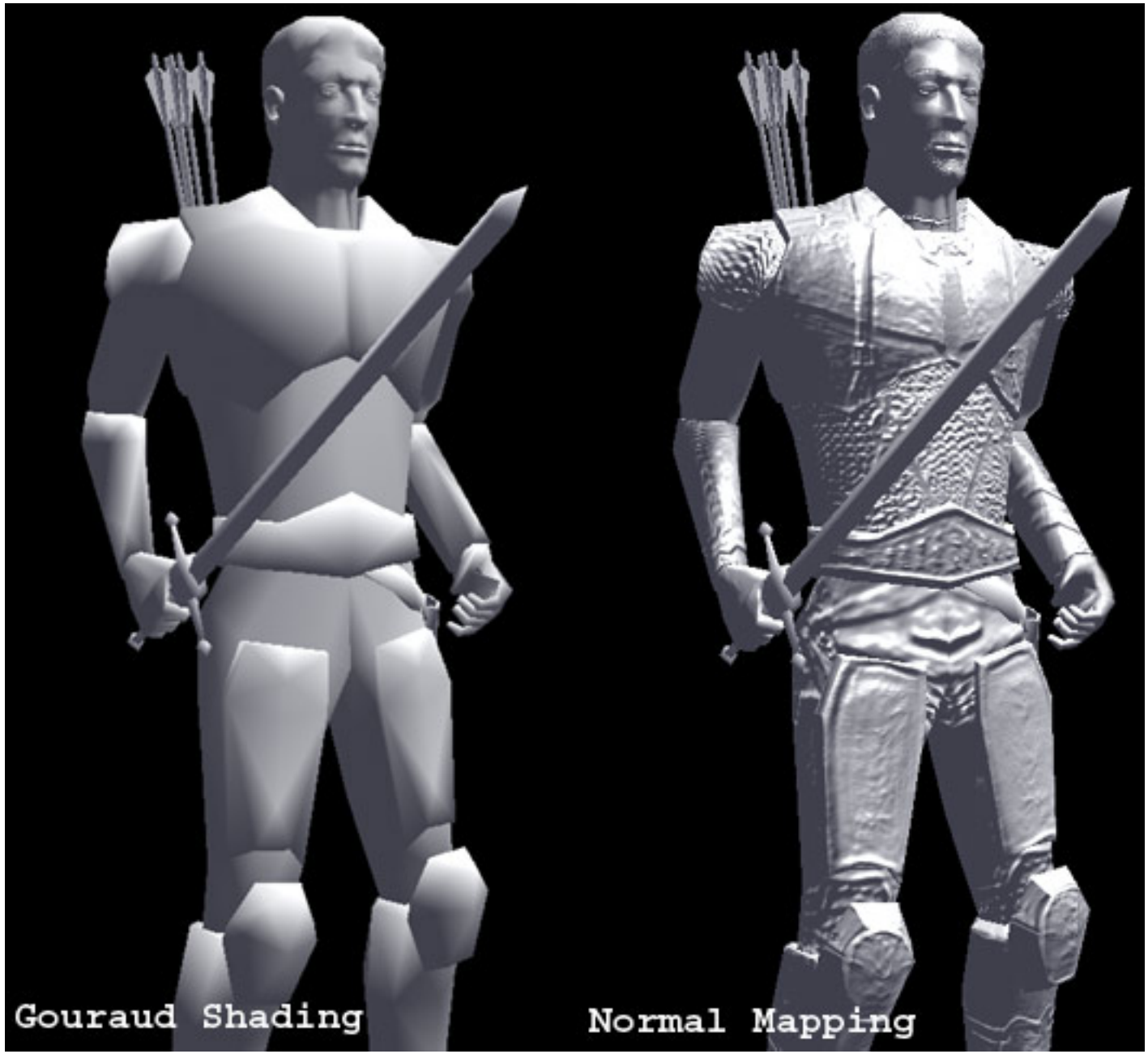
OpenGL - Partikel



OpenGL - Shadow Mapping (Hier mit GLSL)



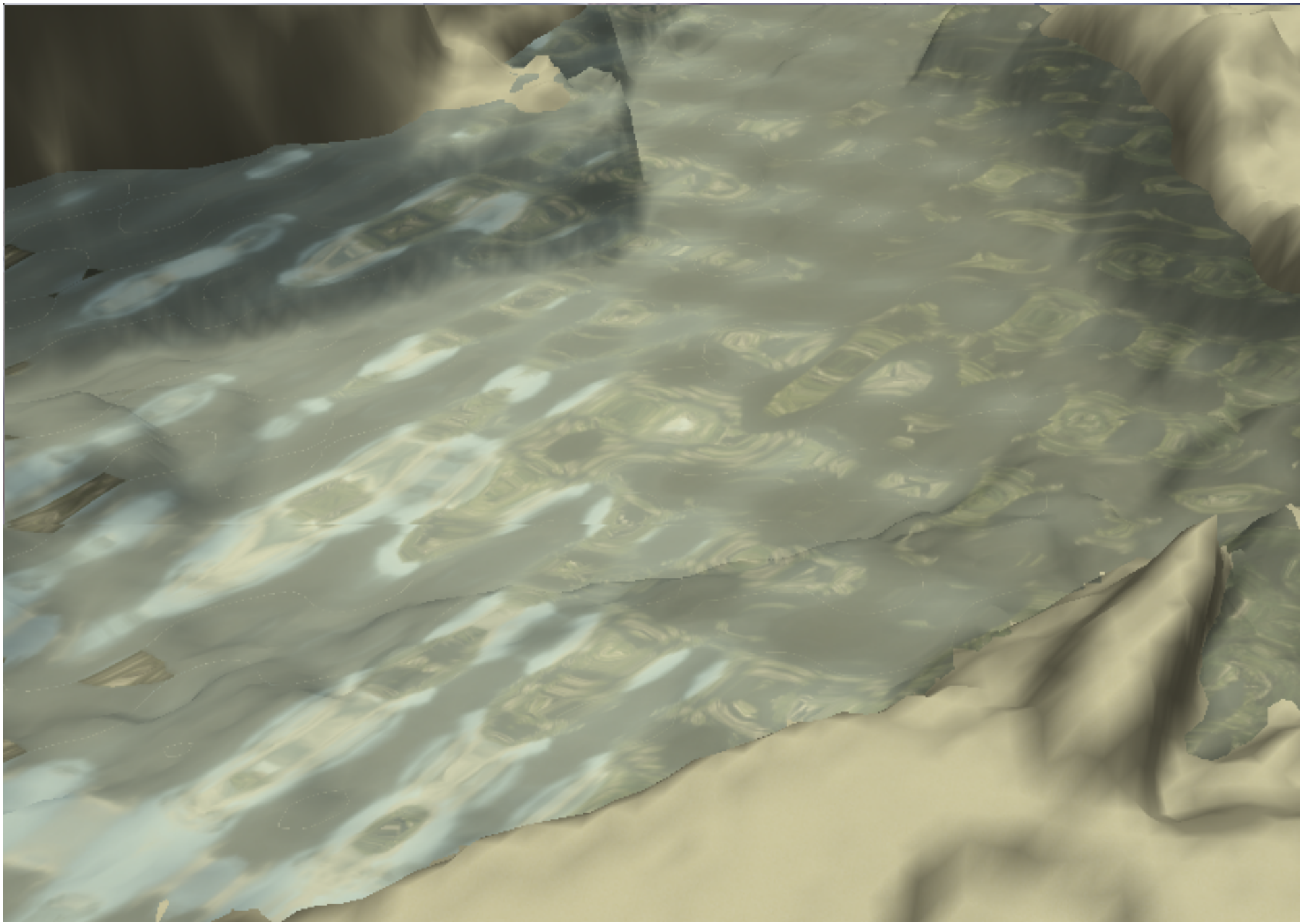
OpenGL - Bump (Normal-) Mapping



OpenGL - (Radial-)Blur Shader

OpenGL - Fog Shader

OpenGL - Wave Shader



OpenGL - HDR Shader (disabled)

OpenGL - HDR Shader (enabled)

Ressources - Linkliste

Shader allgemein

http://nightspawn.com/files/gltut/paper.pdf
http://www.nvidia.com/content/GTC-2010/pdfs/2227_GTC2010.pdf
http://fabiensanglard.net/

GLSL References

Lighthouse 3D - GLSL Shader Management with OpenGL
NEHE GLSL Introduction
The GLSL Orange Book
DGL Shadersammlung

Geometry Shaders

Geometry Shader Tutorial - Hello World
Ein weiteres gutes Geometry Shader Tutorial

Tesselation Shaders

Triangle Tesselation with OpenGL 4.0+
Terrain Tesselation with OpenGL 4.0+