GLSL – GL Shading Language

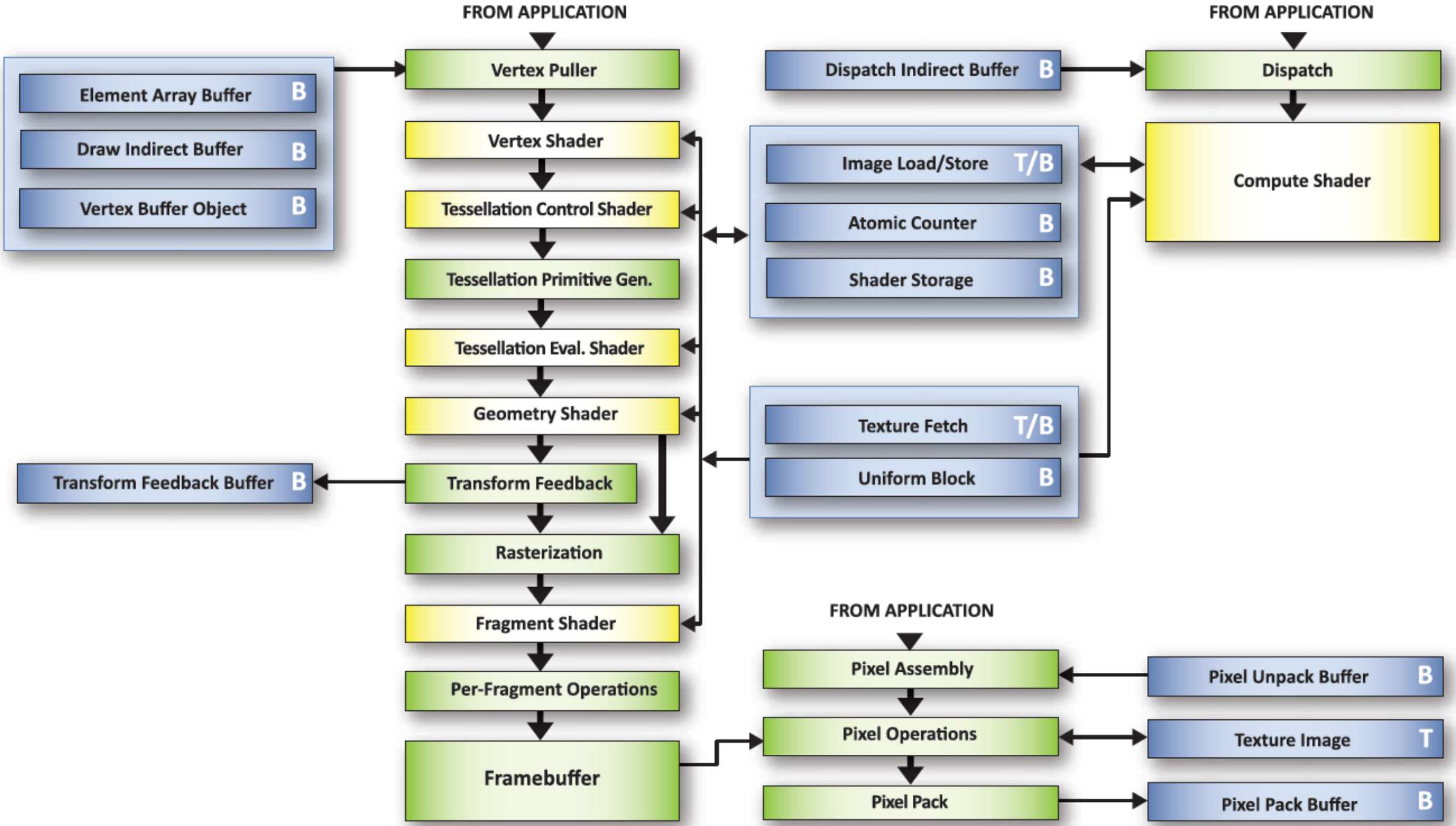
Dr.-Ing. Christoph Fünfzig

#version 330 core

OpenGL Version	GLSL Version
2.0	1.10
2.1	1.20
3.0	1.30
3.1	1.40
3.2	1.50

#extension GL_OES_standard_derivatives:enable

Shader Stufen (OpenGL 4.3)



aus: OpenGL 4.3 Reference Card, Khronos.org

GLSL

- GLSL ist C-ähnliche Sprache für Datentypen

```
bool b;          // Bool-Typ
int i;           // Integer-Typ
float a;         // einfache Genauigkeit
double b;        // doppelte Genauigkeit (selten verwendet!)
vec3 x;          // 3D Vektor mit float-Typ
vec4 y;          // 4D Vektor mit float-Typ
mat3x3 m;        // 3x3 Matrix; auch: mat3 m;
mat4x4 m;        // 4x4 Matrix; auch: mat4 m;
mat2x4 m;        // 2x4 Matrix (selten verwendet!)
```

- keine Casts, nur Konstruktion

```
vec3 coord=vec3(0.0); vec4 origin = vec4(coord, 1);
```

GLSL: Datentypen

- keine Casts, nur Konstruktion

  vec3 coord = vec3(0.0); vec4 o = vec4(coord, 1);

- *Selektoren*

  [0]..[3] Arraydereferenzierung

  .rgba für Farben

  .xyzw für Vertices

  .stpq

  o[0] == o.r == o.x == o.s;

- *Swizzling*

  o.xyz // ergibt vec3 (o.x, o.y, o.z); entspricht vec3(o)

  o.www // ergibt vec3 (o.w);

GLSL: Datentypen

- Es gibt weniger als einfache Genauigkeit in OpenGL ES 2.0

```
float a;          // Einfache Genauigkeit

precision highp   float; // Zusatz für float; default
precision mediump float; // Zusatz für float
```

GLSL: Zu Kontrollstrukturen sage ich nix ☺

- if (..) STMT else STMT;
- int hi;
  switch (hi) {
  case (..):
  STMT;
  }
- while (..) STMT;
  do STMT; while (..);
- for (..; ..;) STMT;

GLSL: Qualifier

- Es gibt keine globalen Variable: Nur lokale und Ein-Ausgabe
- *Qualifier*

```
const              // Konstanten
uniform            // Änderung nur zwischen glDrawArray/glDrawElements
attribute          // in Vertex Shader, Änderung per Vertex
layout (location=1)  //  Zusatz für attribute (Binding Index)
varying            // out Vertex Shader, in Fragment Shader,
                   // Änderung per Fragment/Pixel
```

Beispiele: Vertex Shader, Fragment Shader

- Blatt00.zip
  Keine Attribute, nur lokale Arrays
  Eigene Funktion hsv2rgb
- Blatt06.zip
  Gouraud Shading eines Würfels

GLSL: Vertex Shader, Fragment Shader

```
layout(location=1) in vec4 position;
..
GLuint pos=1; glEnableVertexAttribArray(pos);
glVertexAttribPointer(pos, 3, GL_FLOAT, GL_FALSE, 0, 0)
```

Inputs	in int gl_VertexID; in int gl_InstanceID;

Outputs	out gl_PerVertex {
	vec4 gl_Position;
	float gl_PointSize;
	float gl_ClipDistance[];
	};

https://www.khronos.org/opengl/wiki/Vertex_Shader

GLSL: Vertex Shader, Fragment Shader

layout(location = 3) out vec4 diffuseColor

Inputs	in vec4 gl_FragCoord; in bool gl_FrontFacing; in float gl_ClipDistance[]; in vec2 gl_PointCoord; in int gl_PrimitiveID; in int gl_SampleID; in vec2 gl_SamplePosition; in int gl_SampleMask[]; in int gl_Layer; in int gl_ViewportIndex;

Outputs	out float gl_FragDepth;=gl_FragData.z; // wenn nicht zugewiesen
	out int gl_SampleMask[];

https://www.khronos.org/opengl/wiki/Fragment_Shader

GLSL: Funktionen

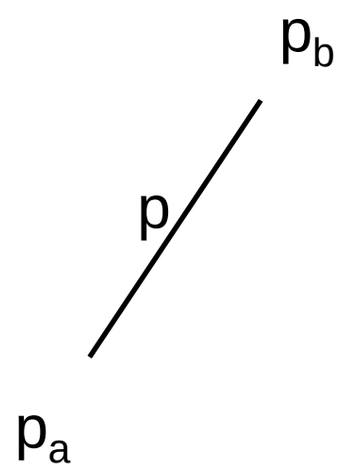
- Viele Bibliotheksfunktionen ☺
  Interpolation, Max, Min, Clamp
  Vektorfunktionen, Matrixfunktionen
  Matrix-Vektor-Multiplikation (sehr ähnlich *glm*)

- Eigene Funktionen
  Rückgabetyp Name (Parameterliste)

```
in        // by-value, wie in C gewohnt
out       // by-reference, Reference wie in C++ gewohnt
inout     // by-reference, Reference wie in C++ gewohnt
return    // wie in C gewohnt
```

Interpolation

- Beispiel: vec4 p(x, y, z, w); *vec4 pa, pb;*

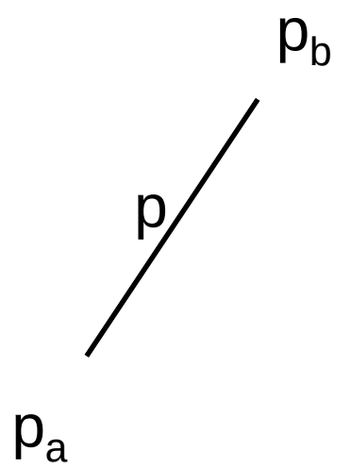


- smooth // Zusatz; default

$$x = \frac{(1-t)x_a + tx_b}{(1-t)w_a + tw_b}$$

Interpolation

- Beispiel: vec4 p(x, y, z, w); *vec4 pa, pb;*

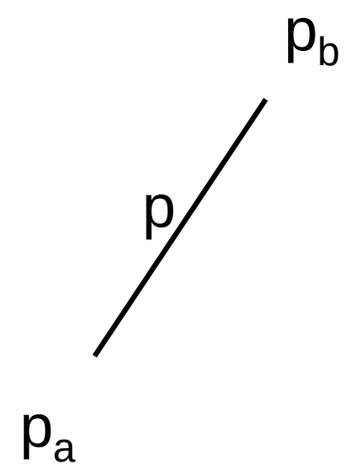


- noperspective // Zusatz; lineare Interpolation

$$x = \frac{(1-t)x_a + tx_b}{1}$$

Interpolation

- Beispiel: vec4 p(x, y, z, w); *vec4 pa, pb;*



- flat // Zusatz; keine Interpolation

$x = x_a$ bzw. x_b

“provoking vertex”

Provoking Vertex

Primitive type	GL_FIRST_VERTEX_CONVENTION	GL_LAST_VERTEX_CONVENTION
GL_POINTS	i	i
GL_LINES	$2i$ - 1	$2i$
GL_LINE_LOOP	i	i + 1, if i < the number of vertices. 1 if i is equal to the number of vertices.
GL_LINE_STRIP	i	i + 1
GL_TRIANGLES	$3i$ - 2	$3i$
GL_TRIANGLE_STRIP	i	i + 2
GL_TRIANGLE_FAN	i + 1	i + 2
GL_LINES_ADJACENCY	$4i$ - 2	$4i$ - 1
GL_LINE_STRIP_ADJACENCY	i + 1	i + 2
GL_TRIANGLES_ADJACENCY	$6i$ - 5	$6i$ - 1
GL_TRIANGLE_STRIP_ADJACENCY	$2i$ - 1	$2i$ + 3

Z-Buffer

Speicherung des Werts $1/z_e$