

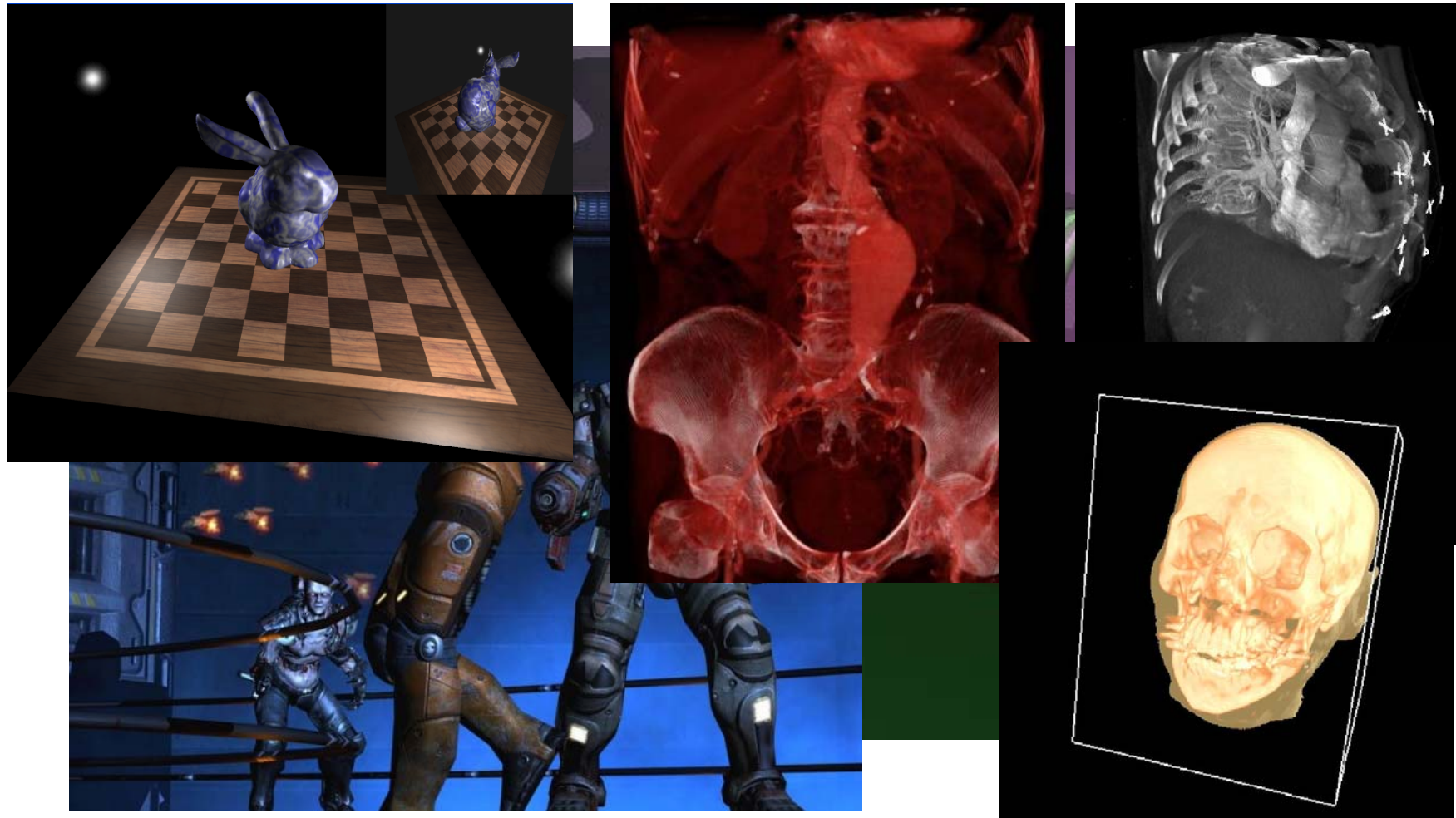
GLSL – GL Shading Language

Oliver Kutter

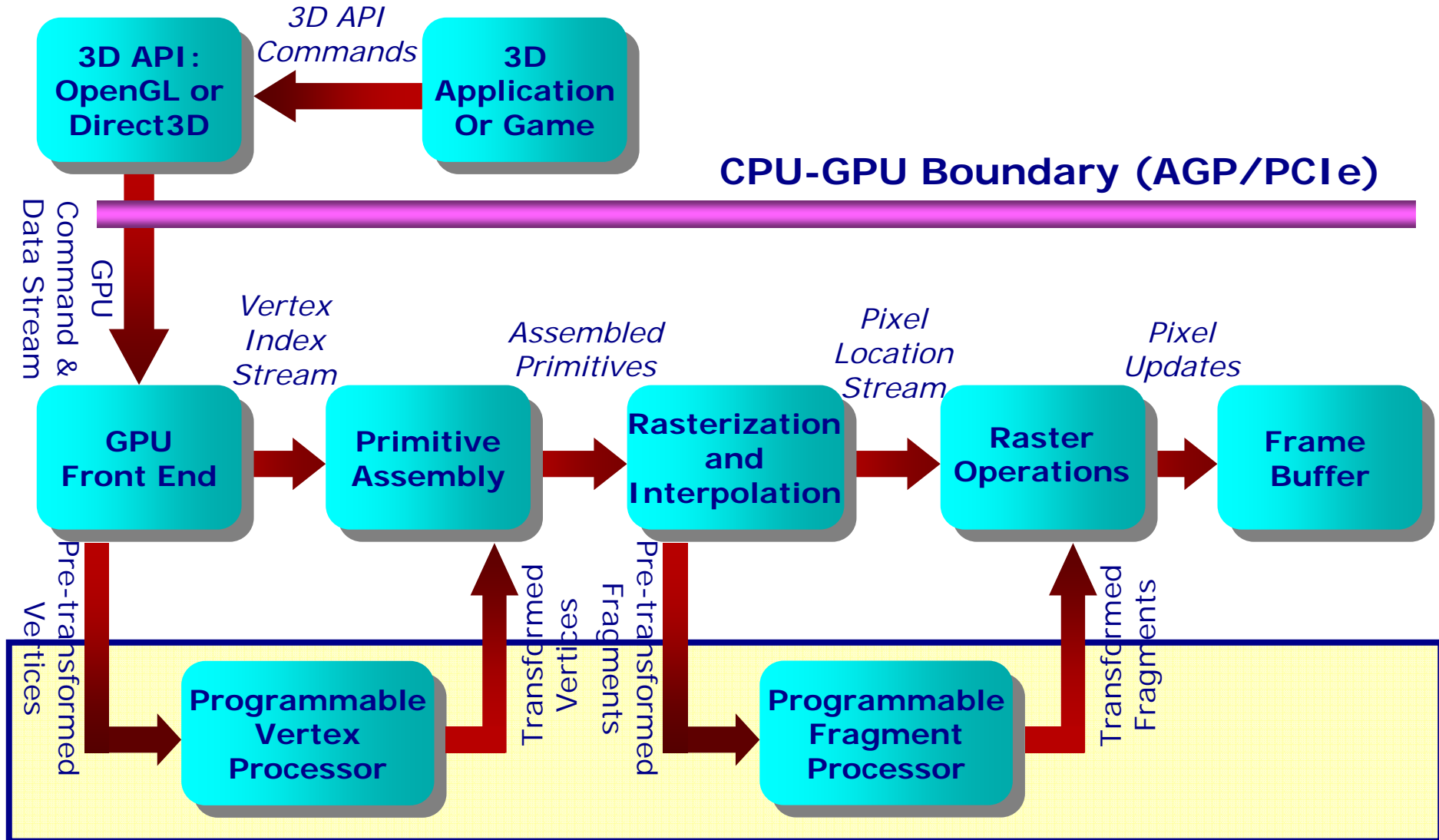
12 June 2007

chair for computer aided medical procedures
department of computer science | technische universität münchen

GPU Programmierung



Programmable pipeline



GLSL – GL Shading Language

Präprozessor Direktiven

#	#error
#define	#pragma optimize(on off)
#undef	#pragma debug(on off)
#if	#line line file
#ifdef	__LINE__
#ifndef	__FILE__
#else	__VERSION__
#elif	
#endif	
	// comment
	/* comment */

Typen – Built in Types

- void
- float vec2 vec3 vec4
- mat2 mat3 mat4
- int ivec2 ivec3 ivec4
- bool bvec2 bvec3 bvec4
- sampler n D samplerCube samplerShadow n D

Typen - Structs

- struct (Mit Einschränkungen u.a.)
 - Keine Qualifier
 - Keine Bit Felder
 - Keine in-place Definitionen
 - Keine anonymen Structs

Typen - Arrays

- arrays (with some minor restrictions)
 - 1D!
 - Grösse muss eine Integer Konstante sein
 - Type und Grösse müssen zusammen definiert werden
 - Jeder Basic Type und Structs
 - Keine Initialisierung bei Deklaration

Typen – Reservierte Typen

- **half hvec2 hvec3 hvec4**
- **fixed fvec2 fvec3 fvec4**
- **double dvec2 dvec3 dvec4**
- **sampler2DRect sampler3DRect**

Funktionen – Scope(s)

- global
 - ausserhalb der Funktionen
 - Gemeinsame globale Variablen müssen gleichen Typ haben
- lokal (nested)
 - Innerhalb Funktionsdefinition
 - Innerhalb Funktion
 - Innerhalb {}

Type Qualifiers

- **const**
- **attribute**
- **uniform**
- **varying**
- **(default)**

Operatoren

- grouping: `()`
- array subscript: `[]`
- function call and constructor: `()`
- field selector and swizzle: `.`
- postfix: `++ --`
- prefix: `++ -- + - !`

Operatoren

- binary: * / + -
- relational: < <= > >=
- equality: == !=
- logical: && ^ ||
- selection: ?:
- assignment: = *= /= += -=

Reservierte Operatoren

- prefix: `~`
- binary: `%`
- bitwise: `<<` `>>` `&` `^` `|`
- assignment: `%=` `<<=` `>>=` `&=` `^=` `|=`

Konstruktoren

- `float()`
- `int()`
- `bool()`
- `vec2()` `vec3()` `vec4()`
- `mat2()` `mat3()` `mat4()`

Beispiele für Konstruktoren

```
float f; int i; bool b;
```

```
float( b)           // b=true?1.0:0.0;  
int( b)            // b=true?1:0;  
bool( b)           // identity  
float( i)          // int to float  
int( i)            // identity  
bool( i)           // i!=0?true:false;  
float( f)          // identity  
int( f)            // float to int  
bool( f)           // f!=0.0?true:false;
```

Konstruktoren für Vektoren

```
vec2 v2; vec3 v3; vec4 v4;
```

```
vec2( 1.0 ,2.0)
```

```
vec3( 0.0 ,0.0 ,1.0)
```

```
vec4( 1.0 ,0.5 ,0.0 ,1.0)
```

```
vec4( 1.0) // all 1.0
```

```
vec4( v2 ,v2) // different...
```

```
vec4( v3 ,1.0) // ...types
```

```
vec2( v4)
```

```
float( v4)
```

Matrix Konstruktoren

```
vec4 v4; mat4 m4;

mat4( 1.0, 2.0, 3.0, 4.0,
      5.0, 6.0, 7.0, 8.0,
      9.0, 10., 11., 12.,
      13., 14., 15., 16.) // row major

mat4( v4, v4, v4, v4)
mat4( 1.0) // identity matrix
mat3( m4) // upper 3x3
vec4( m4) // 1st column
float( m4) // upper 1x1
```

Struct Konstruktoren

```
struct light {  
    float intensity;  
    vec3  position;  
}  
light headLight = light( 0.2  
                        ,vec3( 0.0 ,0.0, 1.0));  
  
// same as  
light headLight;  
headLight.intensity = 0.2;  
headLight.position = vec3( 0.0 ,0.0, 1.0);
```

GL2 Shading Language Components

- component accessor for vectors
 - `xyzw rgba stpq [i]`
- component accessor for matrices
 - `[i] [ij]`

Zugriff auf Vektorkomponenten

```
vec2 v2;
```

```
vec3 v3;
```

```
vec4 v4;
```

```
v2.x // is a float
```

```
v2.z // wrong: component undefined for type
```

```
v4.rgba // is a vec4
```

```
v4.stp // is a vec3
```

```
v4.b // is a float
```

```
v4.xy // is a vec2
```

```
v4.xgp // wrong: mismatched component sets
```

Zugriff auf Vektorkomponenten

```
vec4 v4 = vec4( 1.0, 2.0, 3.0, 4.0 );
```

```
v4.xw = vec2( 5.0, 6.0 ); // (5.0, 2.0, 3.0, 6.0)
```

```
v4.wx = vec2( 7.0, 8.0 ); // (8.0, 2.0, 3.0, 7.0)
```

```
v4.xx = vec2( 9.0, 10.0 ); // wrong: x used twice
```

```
v4.yz = 11.0; // wrong: type mismatch
```

```
v4.yz = vec2( 12.0 ); // (8.0, 12.0, 12.0, 7.0)
```

Array Zugriff

```
vec4 v4 = vec4( 1.0, 2.0, 3.0, 4.0);  
float f;  
int i;  
  
f = v4[0];           // 1.0  
f = v4[3];           // 4.0  
f = v4[4];           // undefined  
// ...  
f = v4[i+1];         // defined for -1<i<3
```

Matrix Zugriff

```
mat4 m4;
```

```
m4[1] = vec4(2.0); // 2nd column = 2.0  
m4[0][0] = 1.0; // upper 1x1 = 1.0  
m4[4][4] = 3.0; // undefined
```

Arrays von Vektoren/Matrizen

```
vec3 v3[2];  
mat3 m3[2];
```

```
v3[0]           // is a vec3  
v3[0][0]       // is a float  
v3[0].x        // is a float  
  
m3[0]          // is a mat3  
m3[0][0]       // is a vec3, 1st column  
m3[0][0][0]    // is a float, upper 1x1  
// :-(
```

GL2 Shading Language

Vector and Matrix Operations

```
mat4 m4,n4;
```

```
vec4 v4
```

```
v4 * m4 // is a vec4
```

```
m4 * n4 // is a mat4
```

Flusskontrolle

- `expression ? trueExpression : falseExpression`
- `if, if-else`
- `for, while, do-while`
- `return, break, continue`
- `discard (fragment only)`

Benutzerdefinierte Funktionen

- Call by value-return
- in out inout keywords
- Function overloading
- Ein Rückgabewert
- Scope Regeln wie in C

Built-in Vertex Special

```
vec4  gl_Position;      // must be written
vec4  gl_ClipPosition; // may be written
float gl_PointSize;    // may be written
```

Built-in Fragment Special

```
bool   gl_FrontFacing;    // may be read
float  gl_FragColor;      // may be written
float  gl_FragDepth;      // may be read/written
float  gl_FragStencil;    // may be read/written
vec4   gl_FragData;       // may be written
vec4   gl_FragCoord;      // may be read
vec4   gl_FragColor;      // may be read
float  gl_FragDepth;      // may be read
float  gl_FragStencil;    // may be read
vec4   gl_FragData;       // may be read
```

Built-in Vertex Attribute

```
// Vertex Attributes  
attribute vec4   gl_Vertex;  
attribute vec3   gl_Normal;  
attribute vec4   gl_Color;  
attribute vec4   gl_SecondaryColor;  
attribute vec4   gl_MultiTexCoordn;  
attribute float  gl_FogCoord;
```

Benutzer definierte Vertex Attribute

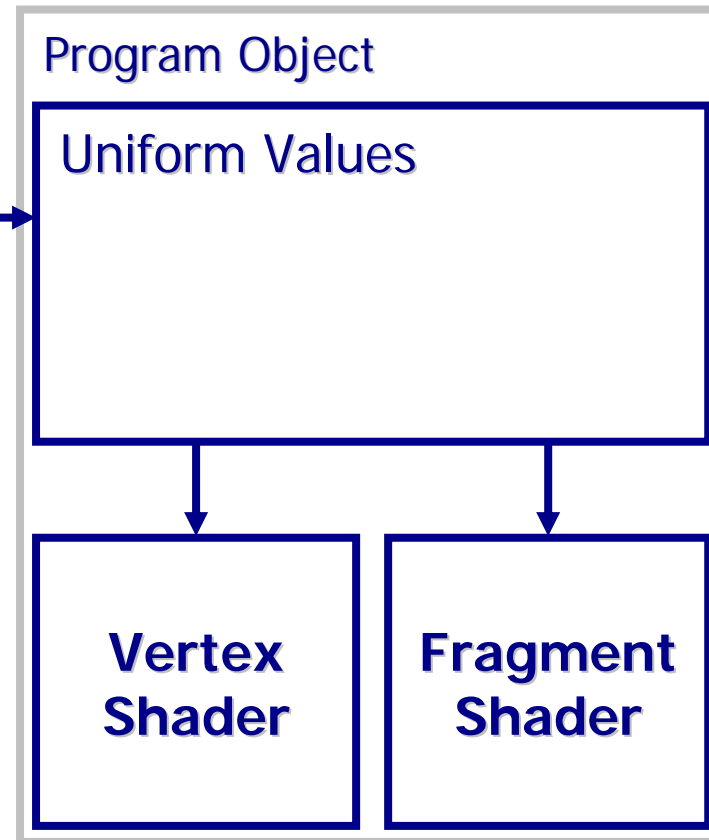
```
attribute vec3  myTangent;  
attribute vec3  myBinormal;  
attribute float myTemperature;  
attribute float myPressure;  
attribute float myRainfall;  
attribute float myTime;  
// etc...
```

Built-in Constants

```
// Implementation dependent constants
// Minimum Maximums
const int gl_MaxLights = 8;
const int gl_MaxClipPlanes = 6;
const int gl_MaxTextureUnits = 2;
const int gl_MaxTextureCoordsARB = 2;
// GL2 Minimum Maximums
const int gl_MaxVertexAttributesGL2 = 16;
const int gl_MaxVertexUniformWordsGL2 = 512;
const int gl_MaxVaryingFloatsGL2 = 32;
const int gl_MaxVertexTextureUnitsGL2 = 1;
const int gl_MaxFragmentTextureUnitsGL2 = 2;
const int gl_MaxFragmentUniformWordsGL2 = 64;
```

GLSL Uniform Variablen

- Built-In Uniforms
 - `gl_ModelViewMatrix`
 - `gl_NormalMatrix;`
 - User defined Uniforms
 - `mTime;`
 - `mScaling;`
- `// etc...`



Built-in Uniforms

```
// Matrix state, p. 31, 32, 37, 39, 40
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ProjectionMatrix;
uniform mat4 gl_ModelViewProjectionMatrix;
uniform mat3 gl_NormalMatrix;
uniform mat4 gl_TextureMatrix[n];
// Normal scaling, p. 33
uniform float gl_NormalScale;
// Depth range in window coordinates, p. 33
struct gl_DepthRangeParameters {
    float near;           // n
    float far;           // f
    float diff;          // f - n
};
uniform gl_DepthRangeParameters gl_DepthRange;
```

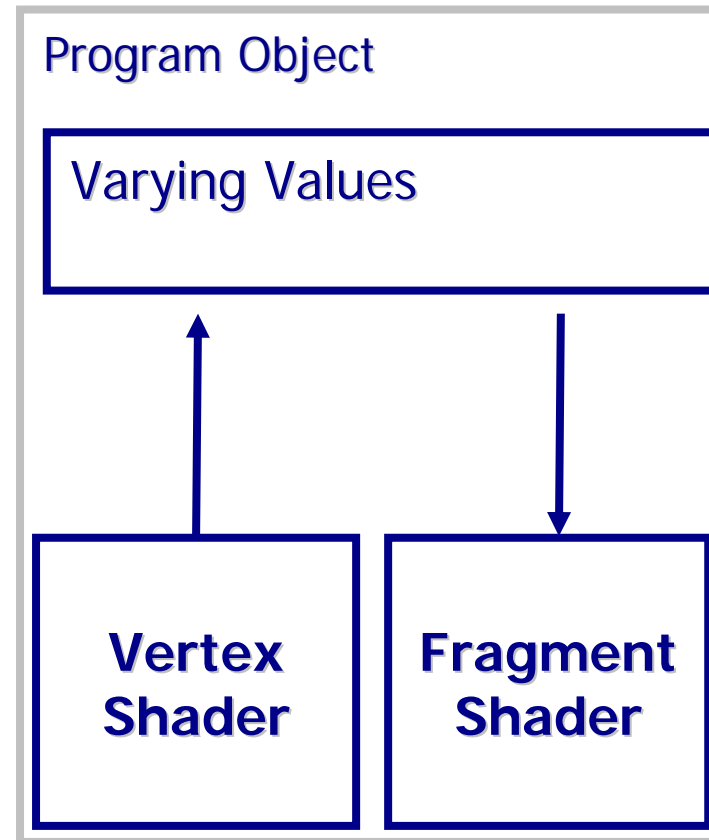
User-defined Uniforms

```
uniform    mat4    myGlobalMatrix;  
uniform    vec4    myAmbient;  
uniform    vec4    myDiffuseMatLight;  
uniform    vec4    mySpecularMatLight;  
uniform    float   myGlow;  
uniform    vec4    myWarm;  
uniform    vec4    myCool;  
uniform    float   myCurrentTime;  
// etc...
```

GLSL Varying Variablen

Variablen die für Kommunikation zwischen Vertex und Fragment Shadern. GL interpoliert automatisch den Wert pro Fragment

(Einsatz z.Bsp. Normalen für per Fragment Lighting)



Built-in Varying Variablen

```
varying    vec4    gl_FrontColor    // vertex
varying    vec4    gl_BackColor;    // vertex
varying    vec4    gl_FrontSecColor; // vertex
varying    vec4    gl_BackSecColor; // vertex

varying    vec4    gl_Color;        // fragment
varying    vec4    gl_SecondaryColor; // fragment

varying    vec4    gl_TexCoord[];   // both
varying    float   gl_FogFragCoord; // both
```

Benutzer definierte Varying Variablen

```
varying  vec3  myNormalPrime;  
varying  vec3  myTangentPrime;  
varying  vec3  myBinormalPrime;  
varying  float myPressurePrime;  
// etc...
```

Built-in Funktionen

```
genType abs( genType x)
genType sign( genType x)
genType floor( genType x)
genType ceil( genType x)
genType fract( genType x)
genType radians( genType degrees)
genType degrees( genType radians)
genType sin( genType angle) // radians
genType cos( genType angle)
genType tan( genType angle)
```

Built-in Texture Funktionen

```
vec4 texture1D( sampler1D sampler  
                ,float coord ,float bias)
```

```
vec4 texture2D( sampler2D sampler  
                ,vec2 coord ,float bias)
```

```
vec4 texture3D( sampler3D sampler  
                ,vec3 coord ,float bias)
```

```
vec4 textureCube( samplerCube sampler  
                 ,vec3 coord ,float bias)
```

Vollständige Funktionen Dokumentation

- <http://www.opengl.org/documentation/glsl/> - GLSL Language Spec
- http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf - GLSL Quick Ref

GLSL Hello World Shaders

GLSL Hello World Vertex Shader

```
// write gl_Position by transforming vertices
// with ModelViewProjection Matrix (MVP)
#define MVP      gl_ModelViewProjectionMatrix
void main( void)           // Vertex Shader
{
    gl_Position = MVP * gl_Vertex;
}
```

GLSL Hello World Fragment Shader

```
// Each fragment is colored in green
// with max opacity
void main( void)           // Fragment Shader
{
    gl_FragColor = vec4( 0.0, 1.0, 0.0 ,1.0);
}
```

Setting up GLSL

Setting up GLSL

- Requirements und Utilities
 - GLSL Program
 - GLSL Program Object
 - GLSL Shader Objects (Vertex + Fragment Shader)
 - GLEW (für GL Extensions, GL > 1.1 unter Windows)

Setting up GLSL Shaders

- Shader erzeugen

```
GLuint vShader = glCreateShaderObject(GL_VERTEX_SHADER);  
GLuint fShader = glCreateShaderObject(GL_FRAGMENT_SHADER);
```

- Shader Source setzen

```
glShaderSource(vShader, 1, vsSource, 0);  
glShaderSource(fShader, 1, fsSource, 0);
```

- Shader kompilieren

```
glCompileShader(vShader);  
glCompileShader(fShader);
```

Setting up a GLSL Program Object

- GLSL Programm erzeugen

```
GLuint glslProg = glCreateProgramObject();
```

- Shader attachen

```
glAttachObject(glslProg, vsShader);  
glAttachObject(glslProg, fsShader);
```

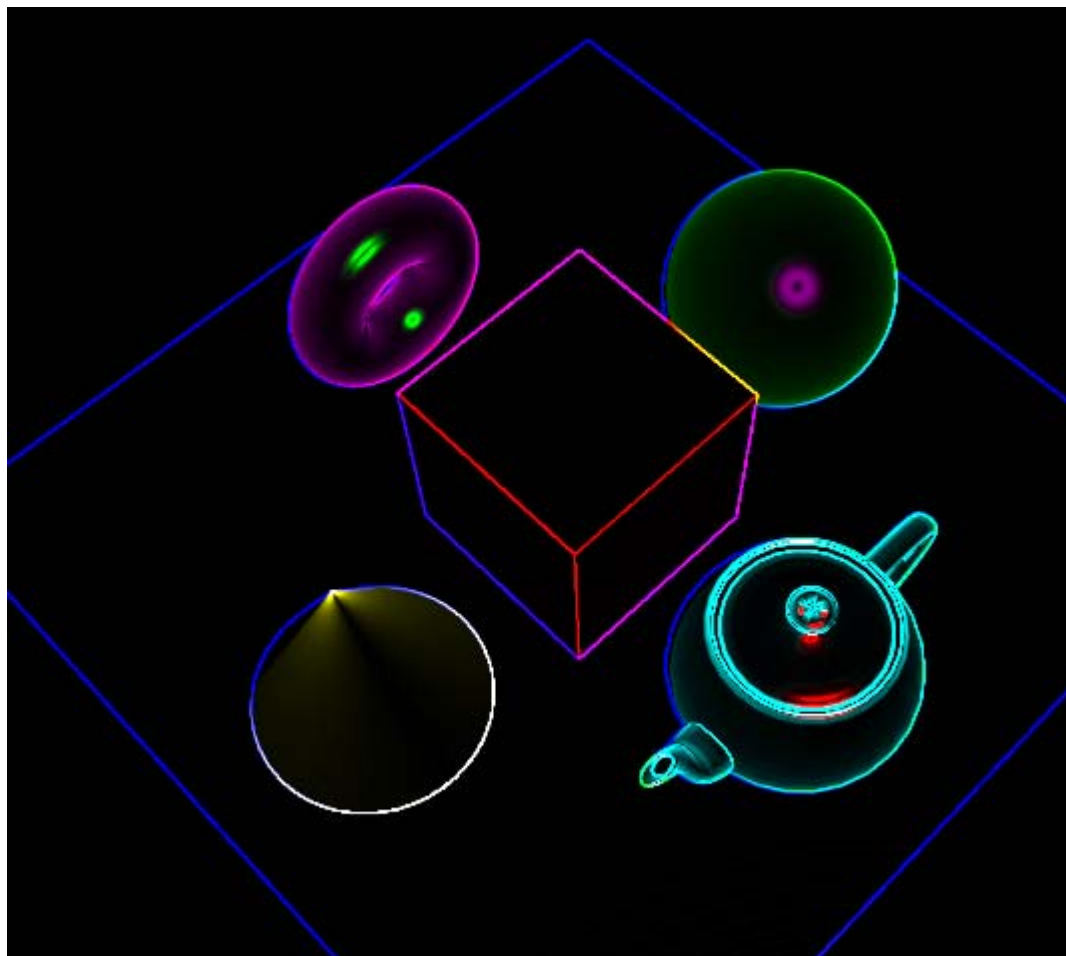
- Programm linken

```
glLinkProgramObject(glslProg);
```

- Programm aktivieren/deaktivieren

```
glUseProgramObject(glslProg); // aktiviere Programm  
glUseProgramObject(0); //
```

Assignment



Assignment - Hinweise

- Zur Bearbeitung wird Ihnen eine objektorientierte Bibliothek für das Programmieren mit GL Shader/Program Objekten sowie ein Anwendungstemplate gestellt, das noch zu ergänzen ist
- Shader Debuggen kann mitunter extrem hart sein → Fügen Sie zum Testen offensichtliche Fehler ein oder schreiben Sie verschiedene Farbwerte als Debug Ausgabe in den Framebuffer

OpenGL Debugging Tips

- Entweder selber Code mit Debug Info anreichern.
 - switch/if/#define etc ... im C/C++ Code
 - Im Fragment Shader z.Bsp das Fragment rot färben wenn Ereignis X eintritt ...
- gDEBugger nutzen (<http://www.gremedy.com/>)
 - Breakpoints in OpenGL setzen
 - OpenGL States/Variablen auslesen
 - Texturen + TexParameter auslesen/anzeigen
 - GPU Profiling Fähigkeiten

References, Books, Tutorials

- <http://www.opengl.org/sdk/>
 - Links to Documentation
 - Links to Tutorials (Lighthouse 3D)
- OpenGL 2.0 Programming Manual (Red Book)
- OpenGL Shading Language (Orange Book)
- Vendor SDKs
 - <http://developer.nvidia.com/page/home.html>
 - <http://ati.amd.com/developer/index.html>

Fragen ?

Bei Fragen und Problemen während der
Bearbeitung der Aufgaben bitte rechtzeitig
melden (kutter@in.tum.de)