Montag 20. Juni 2005

Proseminar: Virtuelle Präsenz

Steve Rechtenbach
steve.rechtenbach@informatik.uni-ulm.de

Inhaltsverzeichnis

Einführung 3

Beispiel 4

Das Architecture Review Board 6

Shader - OpenGL Shading Language 7

Beispiel Shader 9

Quellenverzeichnis 10

Einführung

1992 wurde OpenGL von SGI(Silicon Graphics Inc.) als offener Standart für 2D/3D Programmierung vorgestellt. OpenGL stellt bei genauerem betrachten eine Spezifikation zum Plattform- und Programmiersprachenunabhängigen programmieren von 2D/3D-Computergrafik dar.
Der Befehlssatz von OpenGL besteht heutzutage aus ca. 250 Befehlen welche im Kern festgelegt sind und den Erweiterungen(Extensions), die meist zuerst Hardwarespezifisch von den jeweiligen Grafikkartenherstellern implementiert werden, und erst später gemeinsam Standard bilden.
OpenGL basiert zum größten Teil auf der früheren Arbeit SGI's an ihrer hauseigenen Grafikschnittstelle IRIX-GL für das Unix-Betriebssystem „Irix", wird jedoch heutzutage auf den folgenden Betriebssystemen unterstützt:

AIX
IRIX
FreeBSD
NetBSD
OpenBSD
Linux
Windows 9x
Windows NT
OS/2
Amiga OS
BeOS
MacOs
...und anderen Unix Betriebssystemen...

Diese Breite Unterstützung ist auf die Eigenschaft zurückzuführen, dass OpenGL anerkannter Industriestandard ist, und vollkommen offen entwickelt wird, wodurch die Anpassung für neue Betriebssysteme, aber auch für verschiedene Programmiersprachen relativ schnell zu realisieren ist. Der Kern von OpenGL ist in C programmiert, wodurch die Unterstützung durch diese Programmiersprache selbstverständlich ist. Weitere Sprachen, unter welchen man OpenGL nutzen kann sind somit unter anderem:
C++, Java, Python, Tcl/Tk, Pascal, Delphi, Perl, Scheme, Guile, Haskell, Ada, Fortran, Modula-3.

Beispiel

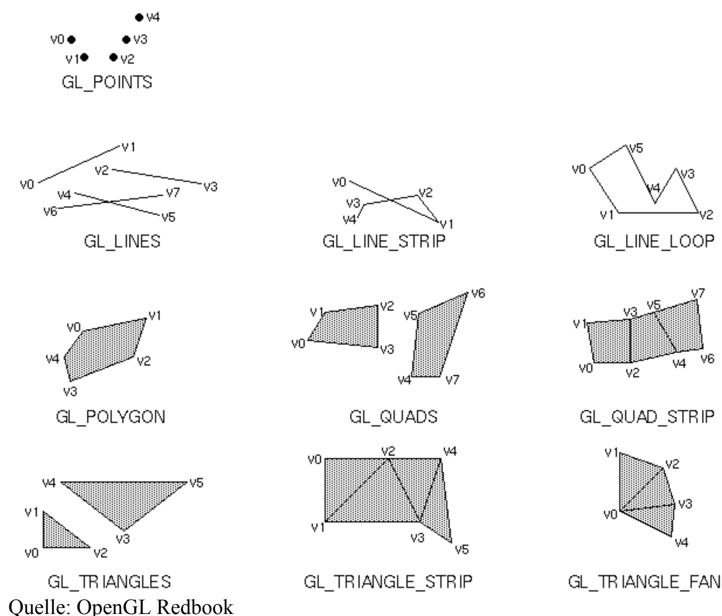
Um OpenGL unter C++ zu benutzen muss man nur die opengl32 linken und im #include Teil die gl.h(Befehle, Funktionen im OpenGL Kern) und die glext.h(alle Extensions) einbinden.

Ein einfaches Beispiel, welches man als Hello-World unter OpenGL bezeichnen könnte, wäre das zeichnen eines geometrischen Objekts.

Am Beispiel eines einfachen Quadrates würde das so aussehen:

```
glBegin(GL_QUADS);
    glVertex3f(-1.0f,-1.0f, -6.0f);
    glVertex3f( 1.0f,-1.0f, -6.0f);
    glVertex3f( 1.0f, 1.0f, -6.0f);
    glVertex3f(-1.0f, 1.0f, -6.0f);
glEnd();
```

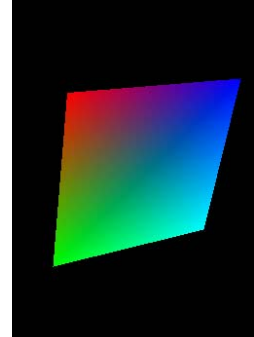
Alle Zeichenvorgänge erfolgen immer zwischen den glBegin, und glEnd Befehlen, wobei in glBegin festgelegt wird, welche Art von geometrischer Figur gezeichnet wird.
In diesem Beispiel ist das ein Viereck, welches als Koordinaten 4 Eckpunkte benötigt.
Unter OpenGL existiert jedoch auch noch eine Vielzahl von anderen Figuren.



Quelle: OpenGL Redbook

Um unserem Beispiel noch ein wenig interessanter zu gestalten kann man es natürlich auch noch im 3-dimensionalen Raum bewegen, und ihm eine Farbe zuweisen.
Dies geschieht durch die folgenden Befehle:

```
glTranslatef(0.0f,0.0f,-6.0f);
glRotatef(45,1.0,1.0,0.0);
glBegin(GL_QUADS);
    glColor3f ( 0.0, 1.0, 0.0);
    glVertex2f(-1.0,-1.0);
    glColor3f ( 0.0, 1.0, 1.0);
    glVertex2f( 1.0,-1.0);
    glColor3f ( 0.0, 0.0, 1.0);
    glVertex2f( 1.0, 1.0);
    glColor3f ( 1.0, 0.0, 0.0);
    glVertex2f(-1.0, 1.0);
glEnd();
```



Was hier passiert ist schon etwas interessanter:
glTranslate
- sorgt für eine Bewegung um 6 Einheiten in das Bild hinein(z-Achse)

glRotate
- dreht das zu zeichnende Objekt um die x- und y-Achse um jeweils 45°

glColor
- gibt jedem Eckpunkt einen RGB Wert

Dieses kleine Beispiel ist natürlich nur einen sehr kleiner Blick auf die extrem umfangreiche API, sollte die einfache Funktionsweise von OpenGL aber zeigen.

Das Architecture Review Board

Da OpenGL als offener Standart entwickelt wurde, gehört es nicht SGI allein, sondern wird von dem ARB(Architecture Review Board), dem „Standarisierungskomitee" der API, verwaltet. Dieses besteht zum einen aus den Promoter-Level Mitgliedern(Stand Juni 05):

IBM, ATI, Dell, Apple, 3Dlabs, Intel, Nvidia, SGI, Sun Microsystems,

die Abstimmungsrecht über neue Spezifikationen betreffend OpenGL haben. Zum anderen besteht das ARB aus den Contributer-Level Mitgliedern zu denen

Evans and Sutherland, Imagination Technologies, Matrox, Quantum3D, S3 Graphics, Spinor GmbH, Tungsten Graphics, and Xi Graphics

gehören.
Das ARB trifft sich alle 3 Monate um zu beraten wir die Entwicklung weitergehen soll, dh. es entscheidet über neue „features" und Erweiterungen des Standards, da er mit der schnellen Grafikhardware Entwicklung Schritt halten muss. Protokolle der Treffen sind unter www.opengl.org frei zugänglich, und können von jedem Interessierten eingesehen werden.
Die Standards in chronologischer Reihenfolge:

1992 OpenGL 1.0
1995 OpenGL 1.1
1998 OpenGL 1.2
2001 OpenGL 1.3
2002 OpenGL 1.4
2003 OpenGL 1.5
2004 OpenGL 2.0

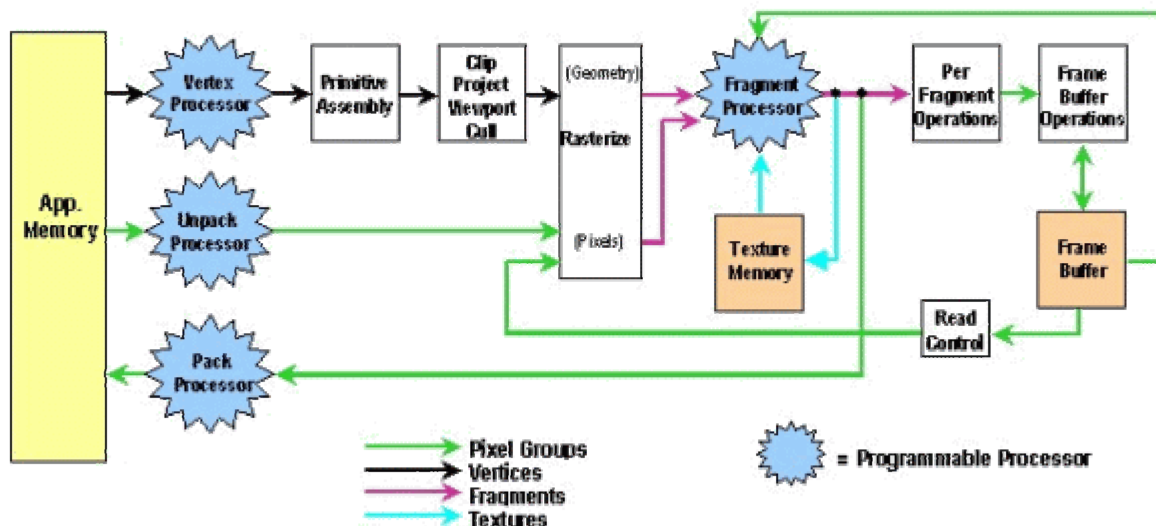
Das schnellere Voranschreiten der Entwicklung in den letzten Jahren, besonders zwischen 2001 und 2004 ist zum Teil auf die rasende Grafikkartenentwicklung, und den Konkurrenzkampf zwischen Nvidia und ATI zurückzuführen, welche sich mit neuen Features gegenseitig auszustechen versuchen. Der zweite Grund, warum bereits 1 Jahr nach OpenGL 1.4 eine neue Version erschien ist die Tatsache, dass mit Version 1.5 die lang erwartete Unterstützung für Shader in die Spezifikation mit aufgenommen wurde.

Shader - OpenGL Shading Language

Die Entwicklung größerer Virtueller Welten(zB. Spielen, Simulationen) setzt einen immer größer werdenden Aufwand in Sachen Komplexität und Detailtreue voraus.

Bilder(von links nach rechts): Stalker, Half-Life², Doom3

Der Trend in Sachen grafischer Programmierung entwickelt sich momentan dahin, dass immer stärker die Notwendigkeit besteht direkten Einfluss auf die Vertex- und Fragmentverarbeitung, welche statisch in der Grafikhardware abläuft, zu nehmen. Die hauptsächlichen Schwachpunkte in der konventionellen Programmierung mit OpenGL lagen darin, dass dies eben nicht möglich war. Die OpenGL Shading Language (GLSL, umgangssprachlich GlSlang) wurde dafür entworfen, um den Anwendungsentwicklern die Möglichkeit zu geben, einen Teil der Rendering Pipeline, um genau zu sein, die oben erwähnten Vertex- und Fragmentverarbeitungseinheiten, zu programmieren.



Quelle: http://graphics.tomshardware.com/graphic/20020222/opengl-04.html

Unabhängig voneinander kompilierbare Einheiten welche unter GLSL geschrieben wurden nennt man Shader. Unter OpenGL existieren zum einen die Vertexshader, welche auf dem Vertex-Processor laufen, und zum anderen die Fragmentshader(ua. auch Pixelshader genannt), welche auf dem Fragment-Processor laufen.

Vertex-Processor:
Der Vertex-Processor verarbeitet die zu ihm geleiteten Vertices samt ihrer Daten. Dazu gehören:

- die Position
- die Normale zu dem Vertex
- die Texturkoordinaten
- die Beleuchtung des Vertex
- und das Material(Reflektionseigenschafen)

Wenn man Vertexshader benutzt, so muss man alle diese Eigenschaften des Vertex, falls man sie benutzen will, in den Shader implementieren.
Mit anderen Worten:
Sobald man einen Vertexshader benutzt, sind die „festen" Verarbeitungsschritte, welche der Vertex-Processor ohne aktivierte VertexShader ausführt, verloren und man muss alle Eigenschaften des Vertex selbst festlegen.
Wenn man das nicht tut, so bleibt im wahrsten sinne des Wortes der Bildschirm schwarz.

Fragment-Processor:
Der Fragment-Processor verarbeitet die zu ihm geleiteten Fragmente samt ihrer Daten: Dazu gehören:

- Operationen auf interpolierten Werten
- Festlegen einer Textur
- Nebel
- Farbe(n)

Analog zu den Vertexshadern muss man alle diese Werte in dem Shader festlegen, um sie anzuzeigen.

Vertexshader sind im Gegensatz zu den Fragmentshadern durch geeignete Treiber seitens der Grafikhardwarehersteller emulierbar. Das bedeutet, dass die GPU nicht zwangsläufig einen programmierbaren Vertex-Processor besitzen muss, man den Shader jedoch voll nutzen kann. Diese Technik ist jedoch um einiges langsamer, da OpenGL nur das Vorhandensein des Vertex-Processors „vorgegaukelt" wird.

Shader haben jedoch nicht nur die bereits erwähnten Vorteile, dass man eine Vielzahl von Features schneller, einfacher und effizienter nutzen kann, sondern sie lösen zum Teil auch die leicht komplizierten Extensions von OpenGL ab. Darunter fallen beispielsweise das Multitexturing, BumpMapping, Schatten und die Beleuchtung. Man kann es jedoch auch exzessiv ausweiten und gar die eigentlichen Zeichenvorgänge nur von Shadern nachbilden lassen.

Beispiel Shader

Zum Abschluss noch ein kleines Bespiel von einem Vertex- und Fragment shader, welche Multitexturing durchführen.
Es sollen auf ein Fragment 4 verschiedene Texturen gelegt werden, welche sich natürich überlagern. Dies wird oft in Spielen eingesetzt, um eine realistische Textur zu erzeugen.
Im OpenGL Quelltext wird anhand des RGBA Farbwertes festgelegt, welche Deckkraft eine Textur hat, zB:
R = 0.5
B = 0.5
G = 0
A = 0
Würde bedeuten, dass die resultierende Textur zu 50% aus Textur1(tex1) und zu 50% aus Textur2(tex2) zusammengesetzt wird. Textur3 und Textur4 tragen zu der resultierenden Textur 0% bei.

Der Vertexshader liest den momentanen Farbwert für den jeweiligen Vertex über die globale Variable gl_Color aus, und schickt ihn über die Variable Color an den Fragmentshader.

Der Fragmentshader empfängt die 4 Texturen aus dem OpenGL Programm, und multipliziert die Texturen jeweils mit dem vom Vertexshader empfangen RGBA Wert.

Vertexshader:

```
varying vec4 Color;
void main(void) {
	//Multitexturing
	Color = gl_Color;
	gl_TexCoord[0].x = gl_MultiTexCoord0.x;
	gl_TexCoord[0].y = gl_MultiTexCoord0.y;
}
```

Fragmentshader

```
uniform sampler2D tex1,tex2,tex3,tex4;
varying vec4 Color;
void main(void) {
	vec4 t1 = texture2D(tex1, gl_TexCoord[0].st);
	vec4 t2 = texture2D(tex2, gl_TexCoord[0].st);
	vec4 t3 = texture2D(tex3, gl_TexCoord[0].st);
	vec4 t4 = texture2D(tex4, gl_TexCoord[0].st);
	gl_FragColor = (t1*Color.r
			+t2*Color.g
			+t3*Color.b
			+t4*Color.a);
}
```

Was hier durch Shader mit relativ wenig Programmieraufwand zu erledigen ist, würde über die konventionelle Extension-Methode ohne Shader 1-2 Seiten Quelltext ergeben, und sogar geringfügig langsamer laufen.

Quellenverzeichnis

OpenGL Logo
http://www.opengl.org/about/logos.html

Overview of OpenGL
http://www.opengl.org/about/overview.html

Architecture Review Board
http://www.opengl.org/about/arb/overview.html

The OpenGL Programming Guide - The Redbook
http://www.opengl.org/documentation/red_book_1.0/

OpenGL Super Bible Third Edition
http://www.starstonesoftware.com/OpenGL/

OpenGL OrangeBook – OpenGL Shading Language
http://3dshaders.com/orangeBook.html

Nvidia
http://www.NVidia.com