OpenGL als API für Augmented und Virtual Reality

Tobias Lang
langt@cip.ifi.lmu.de

Hausarbeit im

Hauptseminar Augmented und Virtual Reality

Inhaltsverzeichnis

1. Einleitung

Die realistische Darstellung unserer Umgebung war schon ein frühes Ziel der Computergrafik und ist es bis heute geblieben. Die Echtzeitberechnung von Formen verbunden mit Licht und Schatten bietet unendliche Möglichkeiten. Angefangen von realistischen Computerspielen über komplexe Visualisierungen bis hin zu Augmented Reality.
Doch noch Anfang der neunziger Jahre konnte anspruchsvolle dreidimensionale Computergrafik nur mittels Hochleistungsrechnern realisiert werden. Erst die rasante Entwicklung der Personal Computer zusammen mit 3D-Beschleunigungskarten brachte den Durchbruch. Heute ist es uns möglich 3D-Grafik ohne teures Equipment zu genießen und zu kreieren.

2. Was ist OpenGL

OpenGL kann als ein Software-Interface für Grafikhardware betrachtet werden. Obwohl es auch ohne spezielle Grafikhardware funktioniert, d.h. von Software emuliert werden kann, ist es doch für den Einsatz mit 3D-Beschleunigern konzipiert. Programmierern wird hierbei ermöglicht, komplexe Algorithmen der Hardware zu überlassen und so Programme portabler und wesentlich performanter zu gestalten.
OpenGL (Open Graphics Library) ist ein Industriestandard der ursprünglich von Silicon Graphics unter dem Namen IrisGL für Ihre gleichnamige Grafikplattform entwickelt wurde. 1992 wurde daraus der offene Standard OpenGL.
Heutzutage betreut ihn das Architecture Review Board (ARB) indem alle großen Grafik-Hardwarehersteller vertreten sind – von 3DLabs bis Sun Microsystems..
Diese passen die API an neue Hardware an, wobei auch Erweiterungen die sog. Extensions möglich sind.
Mittlerweile wurde OpenGL in fast jeder Plattform wie Linux oder Macintosh implementiert und ist damit die einzige plattformunabhängige 3D-Grafik API.
Die gebräuchlichste Programmiersprache ist C++, allerdings existieren verschiedene andere Sprachanbindungen z.B. für Java und Delphi.

3. OpenGL und Windows

OpenGL selbst bietet keinerlei Fensterfunktionen. Stattdessen existieren für jedes Betriebssystem eigene Anbindungen. Unter Windows sind dies die WGL (wiggle) Methoden. Sie erzeugen einen sog. Rendering Context, der an den Device Context von Windows gebunden wird.
Gerade diese, bei GUI-Betriesystemen grundlegenden, Programmieraufgaben wie Fenster erzeugen und ähnliches stellen ein größeres Hindernis für Anfänger dar.
Außerdem geht die grundsätzliche Plattformunabhängigkeit von OpenGL verloren, da sich Linux-Programmierung in diesem Punkt deutlich von Windows unterscheidet. Abhilfe schafft hier das GL Utility Toolkit, kurz GLUT genannt.

Es stellt Methoden für typische Aufgaben wie Message-Management (Message Loop) und Fenstererzeugung bereit. Damit lassen sich zwar sehr schnell OpenGL Programme schreiben, allerdings auf Kosten der Flexibilität (Menüs u.ä.).

4. Grundlagen der 3D Grafik

Um mit OpenGL arbeiten zu können, ist es nicht unbedingt notwendig sich mit der Theorie hinter der 3D Grafik zu beschäftigen. Aber mit Sicherheit schadet ein wenig Hintergrundwissen nicht.

4.1. Warum wir 2D als 3D wahrnehmen

3D Computergrafik wird in meisten Fällen in 2D dargestellt. Ausnahmen bilden nur 3D Stereo Geräte wie Shutterbrillen oder HMDs.
Warum erkennen wir aber in einer 2D Darstellung wie z.B. auf einem Monitor sofort dreidimensionale Strukturen? Die Antwort liegt in unserer Wahrnehmung. Unser Gehirn benutzt nicht nur die Information aus der Bildverschiebung der beiden Augen um Räumlichkeit zu erkennen. Daneben spielen viele andere Faktoren wie Perspektive (entfernte Objekte sind kleiner als nahe), Schattenwurf, Beleuchtung und Bewegung eine große Rolle. All diese Effekte sind auch in 2D ohne Einbußen zu erkennen - der 3D-Eindruck kann also realistisch nachgeahmt werden.
Sollten jedoch in Zukunft geeignete (bzw. billige) Geräte für „echtes“ 3D verfügbar sein, ist dies kein Problem. Der Computer ist leicht in der Lage, für jedes Auge ein eigenes Bild zu berechnen. In modernen Grafikkartentreibern ist dies sogar schon implementiert.

4.2. Lineare Algebra – Vom Normalenvektor bis zum Kreuzprodukt

Die Mathematik spielt in der Computergrafik eine zentrale Rolle. Besonders die Lineare Algebra im R^3 stellt die grundlegenden Konzepte und Verfahren bereit, mit der 3D Grafik erst möglich wird.
In einer 3D-Welt müssen Objekte bzw. deren Geometrien durch geeignete Strukturen repräsentiert werden. Diese sind Skalare, Punkte und Vektoren.
Skalare sind einfache Zahlenwerte.
Punkte (engl. Vertices) geben eine Position im virtuellen Raum an. Sie bestehen aus drei Skalaren für die drei Koordinatenachsen x, y und z.
Vektoren geben wie in der Physik Richtungen an, wichtig z.B. für Lichtberechnungen.
Jetzt zu einigen wichtigen mathematischen Verfahren.

Kreuzprodukt

Oft ist es notwendig zu einer von Punkten aufgespannten Fläche, dem Polygon, denjenigen Vektor zu finden der auf ihr senkrecht steht. Dies geschieht mit dem Kreuzprodukt:

$$C = A \times B = (A_y \cdot B_z - A_z \cdot B_y, A_z \cdot B_x - A_x \cdot B_z, A_x \cdot B_y - A_y \cdot B_x)$$

Das Ergebnis ist genau dieser orthogonale Vektor. Wird er dann noch durch seine Länge geteilt, nennt man ihn den Normalenvektor. Dieser ist besonders für Beleuchtungsberechnungen unverzichtbar.

Skalarprodukt

Bei Lichtberechnungen ist der Winkel zwischen der Lichtquelle und den Polygonen entscheidend, genauer gesagt zwischen der Lichtquelle und dem eben berechneten Normalenvektor. Am einfachsten ist diese Berechnung mit dem Skalarprodukt zu bewerkstelligen.

$$A \bullet B = (A_x \cdot B_x) + (A_y \cdot B_y) + (A_z \cdot B_z) = |A| \cdot |B| \cdot \cos\alpha$$

Dabei ist es oft nicht nötig, den genauen Winkel mit dem inversen Kosinus zu berechnen. Meistens reicht der Wert des Skalarprodukts schon aus, da man daraus die Beziehung zweier Vektoren von -1.0 (entgegengesetzt) über 0.0 (orthogonal) und 1.0 (parallel) ableiten kann.

Matrizen

Matritzen erlauben eine einfache Beschreibung von Transformationen wie Translation, Rotation und Skalierung. Besteht der Wunsch, mehrere Transformationen hintereinander ausführen, kann man die jeweiligen Matrizen multiplizieren und erhält *eine* endgültige Transformationsmatrix. Ein wichtiges Hilfsmittel, besonders für die Laufzeitoptimierung.

4.3. Koordinatensysteme

Der dreidimensionale Raum wird in im Computer durch die drei Koordinatenachsen x, y und z aufgespannt. Neben diesen sog. Welt Koordinaten, durch welches jedes Objekt in der Welt genau positioniert wird, existieren unzählige zusätzliche lokale Koordinatensysteme (Abb. 1)

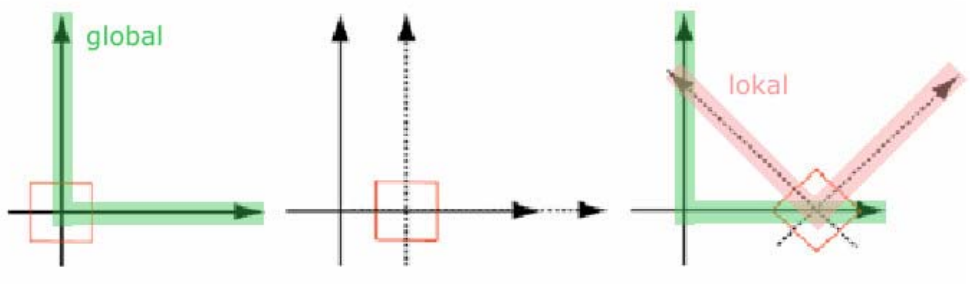

Abb. 1: Globales und lokales Koordinatensystem

4.4. Transformationen

Transformationen erlauben es uns, Gegenstände im Raum zu bewegen, zu drehen und zu manipulieren. Dabei werden grundsätzlich nur die lokalen Koordinatensysteme bewegt, nicht die Objekte selbst. Abb. 1 verdeutlicht dies.
Neben diesen „intuitiven“ Transformationen gibt es noch solche, die den dreidimensionalen Raum in den 2D Raum des Bildschirms transformieren – man spricht in diesem Zusammenhang von Projektionen.

Alles in Allem unterscheidet OpenGL zwischen vier Arten von Transformationen. Zu jeder gibt es eine Analogie aus der Wirklichkeit:

Viewing transformations
: Stativposition und Orientierung der Kamera

Modeling transformations
: Modell bewegen

Projection transformations
: Linse der Kamera einstellen

Viewport transformation
: Das Bild vergrößern/verkleinern

4.5. Primitive

Geometrische Objekte sind in OpenGL durch Punkte repräsentiert. Neben diesen Vertices gibt es noch abgeleitete Primitive wie Linien, Dreiecke, Polygone. Abb. 2 gibt einen kurzen Überblick

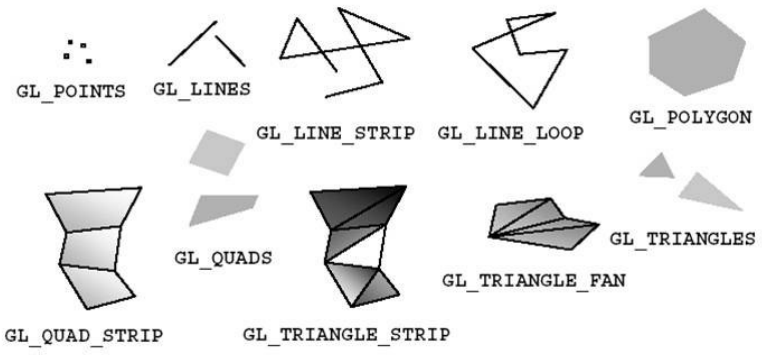

Abb. 2: Primitive in OpenGL

5. Architektur von OpenGL

Das wichtigste Konzept von OpenGL ist die State-Machine. Sie besteht aus Hunderten von Attributen die das Rendering steuern. Angefangen von Rendering-Stil (z.B. wireframe oder solid) über Beleuchtung bis zum Texturing. Immer wenn ein Vertex gezeichnet wird, hängt seine letztendliche Darstellung von diesen Attributen ab. Zur Manipulation der State-Machine werden spezielle Funktionen (state functions) benutzt.

6. OpenGL im Einsatz

6.1. Vorbereitungen

OpenGL ist in erster Linie für den Einsatz mit C++ konzipiert, es wurden aber mit der Zeit auch Anbindungen für andere Sprachen entwickelt. Jedoch ist C++ in dem meisten Fällen die erste Wahl, hauptsächlich wegen seiner überlegenen Performanz.
Um ein OpenGL Programm in C++ kompilieren zu können, müssen lediglich zwei Headerdateien GL.H und GLU.H eingebunden werden. Zusätzlich werden noch die beiden Bibliotheken opengl32.lib und glu32.lib gelinkt (unter Windows). Anschließend kann man die Möglichkeiten von OpenGL vollständig nutzen.

6.2. Grundkörper erstellen

Jedes geometrische Objekt wird in OpenGL mittels Punkten beschrieben. Da diese Punkte aber unterschiedlich verknüpft werden können, z.B. zu Linien oder Polygonen, spezifiziert man vorher die Primitvart.
Das folgende Codebeispiel zeichnet ein Dreieck.

```
glBegin(GL_TRIANGLES);
  glVertex3f(0.0f,0.0f,0.0f);
  glVertex3f(1.0f,0.0f,0.0f);
  glVertex3f(0.0f,1.0f,1.0f);
glEnd();
```

6.3. Transformationen

Transformationen können verschachtelt werden, indem man die Matrix Stacks benutzt. Dabei wird eine aktuelle Transformationsmatrix mit *glPopMatrix()* geladen und mit *glPushMatrix()* wieder auf den Stack gelegt.
Wie schon erwähnt gibt es in der 3D Grafik eine ganze Reihe von Transformationsarten.
Um zu spezifizieren welche man gerade anwenden will, gibt es den Befehl *glMatrixMode(...).* Das folgende Codebeispiel zeichnet einen Quader mit Kugel und zeigt den typischen Aufbau einer Transformation.

```
void render() {
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glPushMatrix();
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(-5.0f,1.0f,0.0f,0.0f);
    DrawCube();
  glPopMatrix();
  glPushMatrix();
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(15.0f,1.0f,0.0f,0.0f);
    DrawSphere();
  glPopMatrix();
 }
```

6.4. Beleuchtung

Lichtberechnung ist unverzichtbar möchte man ein annähernd realistisches Ergebnis haben. Allerdings wäre eine genaue Simulation des natürlichen Lichts viel zu aufwendig um in Echtzeit durchgeführt werden zu können. Deswegen behilft man sich mit stark vereinfachten Beleuchtungsmodellen. In OpenGL ist dies das Phong-Modell. Es berechnet für jeden Vertex eine Lichtintensität und interpoliert diese über die angrenzenden Flächen. Diese Methode ist sehr schnell, kommt aber bei Modellen mit sehr wenigen Polygonen schnell an seine Grenzen und wirkt unrealistisch. Abhilfe bieten die neuen Hardwarebeschleuniger mit ihren Pixel- und Vertexshadern mit denen auch realistischere Beleuchtungs-modelle realisiert werden können.
Die endgültige Erscheinung eines Objekts wird nicht nur durch die Lichtquelle bestimmt sondern auch durch seine eigenen Materialeigenschaften. Diese setzen sich in OpenGL aus verschiedenen Komponenten wie Grundfarbe (diffuse light), Glanzfarbe (specular light) und emittierende Farbe zusammen.

6.5. Texture Mapping

Erst durch das „überziehen“ von Polygonen mit Bitmapdaten lassen sich viele Effekte ohne große Ressourcen verwirklichen. Das zuweisen von Texturen ist in OpenGL recht einfach. Man muß lediglich für jeden Vertex die Texturkoordinaten angeben und die entsprechenden Einstellungen der State-Machine vornehmen.

7. Verwendung für Augmented Reality

OpenGL ist plattformunabhängig und ein offener Standard. Ideale Vorraussetzungen um im universitären Bereich eingesetzt zu werden, in dem Augmented Reality Anwendungen hauptsächlich entwickelt werden.
Die konkurrierende API DirectX von Microsoft bietet jedoch Möglichkeiten, die über OpenGL hinausgehen – insbesondere die Unterstützung für Eingabegeräte und Medienstreams wie Video.
OpenGL ist aber mit Sicherheit einfacher zu erlernen und allein deswegen perfekt geeignet um den Einstieg in die Echtzeit Grafik zu wagen. Zusätzlich wird OpenGL vom ARToolkit, der Standardbibliothek für AR, unterstützt.

8. Ausblick

Der OpenGL Standard wird leider nur langsam aktualisiert. In den zwölf Jahren seines Bestehens gab es gerade mal drei Updates – die neueste Version 2.0 wurde gerade auf der SIGGRAPH 2004 vorgestellt. Zwar lassen sich neue Hardwaremöglichkeiten mit den sog. Extensions ausnützen, diese sind aber unübersichtlich und oft nicht ausreichend genormt. Daneben wird DirectX rasant weiterentwickelt und nähert sich mit jeder Version der Einfachheit von OpenGL an.
OpenGL wird jedoch mit Sicherheit nichts von seiner Popularität Einbüßen. Seine Plattformunabhängigkeit macht es unverzichtbar, gerade im Hinblick auf das sich immer weiter verbreitende Linux. Allerdings kann es durchaus sein, das sich die modernsten Effekte zuerst nur mit DirectX verwirklichen lassen. Gerade im Spielesektor könnte daher OpenGL auf Dauer das Nachsehen haben...

Literaturverzeichnis

Kevin Hawkings, Dave Astle, André LaMothe, OpenGL Game Programming, 2001, Prima Publishing, California

http://www.opengl.org

http://nehe.gamedev.net

http://www.opengl.org/resources/tutorials/s2001/notes/opengl.pdf