Courses 3DCx, C1: OpenGL, FAQs

(teilweise in Anlehnung an Woo,Neider,Davis,Shreiner: OpenGL Programming Guide, Third Ed. AddisonWesley 1999)

Was ist OpenGL ?

OpenGL = Open Graphic Language wurde ab 1992 von Silicon Graphics Inc. zunächst für IRIS Workstations entwickelt und ist inzwischen standardisiert vom OpenGL Architecture Review Board (ARB mit den Mitgliedern SGI, Compaq, IBM, Intel, Microsoft, HP, Evans&Sutherland, Intergraph).
OpenGL ist keine Sprache sondern eine C++Bibliothek bestehend aus ca. 200 Funktionen für 3D. Es besitzt keine Kommandooberfläche (diese kommt von Windows, Linux, MacOS oder einer primitiven OpenGLAuxiliary Library glaux.lib) und keine Beschreibung für 3D-Modelle (die bietet OpenInventor). Es besitzt nur "geometric primitives" wie points, lines, polygones, images, bitmaps. OpenGL leistet:
1. Zusammensetzen von "geometric primitives"
2. Anordnen im Raum vor dem Beobachter und der Lichtquelle
3. Füllen der Flächen mit Farben und Bildern = Rendering, Texturing
4. Berechnen der Pixel für den Monitor = Rasterization

Client-Server-Architektur:
Der OpenGL-Client ist ein Anwendungsprogramm, das OpenGL-Befehle enthält.
Der OpenGL-Server ist betriebssystemunabhängig ein anderer Rechner im Netzwerk oder die eigenen Graphikkarte oder die eigene CPU, wenn die Graphikkarte OpenGL nicht unterstützt. Die Befehle des Clients werden in jedem Fall umgewandelt in ein plattformunabhängiges Format (das sogennnte OpenGL-Protokoll) welches logisch und physikalisch über Netzwerke und Plattformen hinweg transportiert (der Weg kann auch ganz kurz nur zur eigenen CPU führen) und am Ziel (beim Server) als Monitorgraphik ausgeführt wird.
OpenGl wurde entwickelt für Graphikcomputer, d.h. für Maschinen, deren Rechenleistung in der Graphikkarte steckt. Wenn weder Server noch OpenGL-Graphikkarte vorhanden ist, dann implementiert Windows OpenGL ohne Hardware nur per Software (generic = software only).

Erweiterungen von OpenGL:
1. OpenInventor ist eine C++ Klassenbibliothek von Silicon Graphics, die auf OpenGL aufbaut und eine Szenen-Beschreibungssprache für den Entwurf und das Speichern komplexer 3D-Szenen enthält.
2. VRML = Virtual Reality Markup Language ist eine auf OpenGL (und OpenInventor) aufbauende Scriptsprache für das Internet, die von NetscapeNavigator und InternetExplorer interpretiert wird.
3. DirectDraw und Direct3D sind Abkömmlinge und Konkurrenten von OpenGL. Diese Bibliotheken von Microsoft umgehen alle Schichten des Betriebssystems. Sie ersetzen GDI, Client-Server-Architektur und Grapikkartentreiber durch einen dünnen Hardware Abstraction Layer HAL. Wo keine HAL-Unterstützung vorhanden ist, wird ein Softwarezugriff über HEL = Hardware Emulation Layer versucht, wenn auch dieser Zugriff fehlschlägt, wird nach wie vor über das Windows-GDI gearbeitet. Man kann unter Windows98 und Windows2000 (nicht einsetzbar unter NT!) eine DOS-ähnliche Geschwindigkeit der Zugriffe auf die Graphikkarte erreichen. Diese Bibliotheken vereinen die Ideen von OpenGL mit einem sehr schlanken Treibermodell. Die C++Bibliotheken DirectDraw, Direct3D und DirectSound (Sammelbegriff: DirectX) sind die Basis der aufwendigen und schnellen PC-Games. Übrigends kann man OpenGL und Direct3D parallel benutzen. Beliebige Mischungen sind möglich.
see also: http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/directx/dxintro_6d0v.htm

Was ist ein Rendering Context ?

OpenGL zeichnet nicht direkt in den aktuellen ClientDC, wie etwa `dc.LineTo` (das wäre plattformabhängig). OpenGL zeichnet in eine eigene vom Betriebssystem unabhängige Datenstruktur, den OpenGL Rendering Context GLRC. Der GLRC muss natürlich ungefähr die gleichen Daten enthalten wie der DC, aber in einer Form, dass die OpenGL-Befehle unter Linux, MaxOS etc. identisch aussehen. Vor dem ersten OpenGL-Befehl muss also ein GLRC im Speicher angelegt werden. Das macht man am besten einmal beim Start in `CChildView::OnCreate(...)`:

(1) `CClientDC dc( this );`

(2)
```
PIXELFORMATDESCRIPTOR pfd = {
   sizeof( PIXELFORMATDESCRIPTOR ),
   1,
   PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
   PFD_TYPE_RGBA,
   24,0,0,0,0,0,0,0,0,0,0,0,0,0,32,0,0,
   PFD_MAIN_PLANE,0,0,0 };
```

(3) `int i = ChoosePixelFormat( dc.m_hDC, &pfd );`

(4) `SetPixelFormat( dc.m_hDC, i, &pfd );`

(5) `HGLRC hglrc = wglCreateContext( dc.m_hDC );`

(6) `wglMakeCurrent( dc.m_hDC, hglrc );`

Im Klartext:

(1) Reserviere im Speicher einen aktuellen DC der nutzbaren Fläche (Innenbereich ohne Kopfbalken und ohne Rand) des gerade am Bildschirm aktiven Fensters unter dem Namen dc. Aus der Datenstruktur dc kann man in (3), (5) und (6) eine Kennung (=handle) `dc.m_hDC` auslesen, die wie eine Hausnummer den dc identifiziert.

(2) Fülle eine Datenstruktur `PIXELFORMATDESCRIPTOR` mit Wunschwerten wie OpenGL den DC am liebsten haben würde (z.B. mit DoubleBuffer, mit 24 Bit pro Pixel = True Color, etc.)

(3) `ChoosePixelFormat` untersucht, ob und wie weit der aktuelle DC diesen Wünschen entsprechen kann und gibt eine Versionsnummer i zurück, die am besten passt.

(4) `SetPixelFormat` greift physikalisch auf die Graphikkarte, stellt diese so um, dass der für OpenGL beste DC abgebildet werden kann und schreibt die jetzt tatsächlich vorhandenen Eigenschaften (die so nah es geht den Wünschen entsprechen, aber mit denen nicht identisch sein müssen) zurück nach `PIXELFORMATDESCRIPTOR` mit Hilfe des Pointers `&pfd`.

(5) Nachdem der DC nun Wunschform hat, reserviert `wglCreateContext` mit diesem Wissen eine Datenstruktur GLRC, die weiß, was die Hardware kann und die trotzdem plattformunabhängig ist. `hglrc` ist dessen Kennung (=handle).

Cave: Man muss den GLRC löschen, wenn man ihn nicht mehr braucht: `wglDeleteContext( hglrc );` (was im Musterprogramm aus Schlankheitsgründen unterlassen wird).

(6) Der DC ändert sich häufig (Bufferwechsel, Verschiebungen, Überlagerung etc.). Diese Änderungen müssen dem GLRC mitgeteilt werden durch `wglMakeCurrent( dc.m_hDC, hglrc );`, ohne dass man GLRC neu erzeugen müßte.

`wglMakeCurrent( dc.m_hDC, hglrc );` steht im Musterprogramm in der `CChildView::OnTimer()`-Funktion unmittelbar vor den ersten OpenGL-Befehlen.

Was ist ein Double Buffer ?

Animationen sind Bildwechsel, die so schnell sind, dass eine Bewegungsillusion entsteht. In Kino und Fernsehen verschwindet Bild i und wird durch Bild i+1 ersetzt. Bei animierten Graphiken gibt es aber nur ein einziges Bild und Graphik i verschwindet nicht, wenn Graphik i+1 gemalt wird, sondern man sieht i, i+1,...,i+n (siehe Übung 3 anim1.exe). Will man i löschen, dann muss man den gesamten Bildspeicher durch Hintergrund neu füllen. Die Bewegungsillusion geht dabei verloren, das Bild flickert. Ausweg: Man benutzt zwei Bilder, d.h. zwei Bildspeicher, GL_FRONT wird gezeigt, und GL_BACK wird parallel gelöscht und neu gezeichnet. Unmittelbar gezeichnet wird also immer nach `glDrawBuffer( GL_BACK )`. Ist GL_BACK fertig, macht man GL_BACK zu GL_FRONT und damit sichtbar und GL_FRONT zu GL_BACK und unsichtbar mit Hilfe der API-Funktion `SwapBuffers( dc.m_hDC );`.

Vorteil: Kein Flickern, Nachteil: doppelter Speicherplatz in der Graphikkarte notwendig.

Die Eigenschaft, dass das Videomemory in zwei symmetrische GL_FRONT- und GL_BACK-Hälften geteilt sein soll, muss man vorher anmelden im 3. Wort des `PIXELFORMATDESCRIPTOR` mit dem flag `PFD_DOUBLEBUFFER`

Was bedeutet glMatrixMode(GL_MODELVIEW) ?

Scroll, Zoom, Rot, Scherung formuliert OpenGL in Form einer 4x4 Matrix mit 16 float-Zahlen. Scroll, Zoom, Rot, Scherung kann sich beziehen auf das Graphikmodell, auf den Betrachter, auf die Perspektive. Jeder dieser Bezüge hat seine eigene Matrix.
`glMatrixMode( GL_MODELVIEW )` legt fest, dass mit dem nachfolgenden Befehl `glRotatef( 2.0f, 1.f, 1.f, 1.f );` das Grapikmodell gedreht werden soll und nicht der Betrachter und nicht die Perspektive.

Was bedeutet glRotatef(2.0f, 1.f, 1.f, 1.f); ?

glRotate ist der Befehl zur Drehung. Das kleine f am Ende des Befehlsnamens bedeutet, dass alle Parameter vom Datentyp float sein sollen. Obwohl der C-Compiler alle Gleitkommazahlen ohne kleines f automatisch als double behandelt, rechnet sich mit floats merklich schneller als mit doubles und floats benötigen nur den halben Speicherplatz von doubles. Der erste Parameter ist der Drehwinkel 2.0f Grad. Das kleine f hinter 2.0 sagt dem Compiler, dass das float-Konstante ist (2.0 ohne f würde er als double-Konstante behandeln, Nullen vor oder hinter dem Punkt kann man weglassen: 0.0f = 0.f = .0f). Erproben Sie Verkleinerungen (die Animation wird langsamer) und Vergrößerungen (die Animation wird schneller).
Der 2. Parameter bedeutet eine Drehung um die x-Achse (diese Achse geht horizontal durch die Mitte des Fensters).
Der 3. Parameter bedeutet eine Drehung um die y-Achse (diese Achse geht vertikal durch die Mitte des Fensters).
Der 4. Parameter bedeutet eine Drehung um die z-Achse (diese Achse geht senkrecht in die Mitte des Fensters).
Da um alle 3 Achsen gleichzeitig gedreht wird ist, verläuft die Bewegung kompliziert um einen Summenvektor mit 45 Grad Winkel zu allen 3 Raumachsen.
Experimentieren Sie mit diesen Parametern. Beispiel: Setzen Sie x- und y-Drehung auf 0.0f. dann sehen Sie eine einfache planare Drehung in 2D.

Was bedeuten glColor3f und glClearColor ?

`glColor3f( 0.f, 1.f, 0.f);` setzt die Malfarbe = Vordergrundfarbe auf grün.
Beispiele: `( 0.f, 0.f, 0.f )` = schwarz, `( 1.f, 0.f, 0.f )` = rot
`glClearColor( 1.f, 1.f, 1.f, 0.f );` setzt die Löschfarbe = Hintergrundfarbe auf weiß.
Der 4. Parameter löscht den DepthBuffer und hat hier keine Bedeutung.
Warnung: Wenn Vordergundfarbe und Hintergrundfarbe identisch sind, ist die Animation nicht zu sehen und der Fehler schwer zu finden.

Was bedeutet GL_COLOR_BUFFER_BIT ?

OpenGL verwaltet 4 Buffers: Color, Depth, Stencil, Accumulation. Alle Buffer sind als Bildmatrizen organisiert und haben identische Spalten- und Zeilenzahl. Jedes Pixel steht in jedem der 4 Buffer am immer gleichen Platz. Man kann die Buffer einzeln löschen durch:
glClear(GL_COLOR_BUFFER_BIT);
glClear(GL_DEPTH_BUFFER_BIT);
glClear(GL_STENCIL_BUFFER_BIT);
glClear(GL_ACCUM_BUFFER_BIT);

Was ist ein GL_LINE_LOOP ?

Mit
`glBegin( GL_LINE_LOOP )`
Punkt**0**; Punkt**1**; Punkt**2**; ...Punkt**i**;....Punkt**n-1**;
`glEnd();`
zeichnet man ein geschlossenes Polygon. Erproben Sie auch `GL_LINE_STRIP`, `GL_LINES` und `GL_POINTS`.
Unsere Punkte habe die Form von 3D-float-Koordinaten und werden an OpenGL einzeln übergeben in der Form `glVertex3f( x, y, z );` wobei die x,y,z im Zahlenbereich zwischen -1.f und +1.f liegen und aus unserem Array `FPoint f[nMax]` entnommen werden.

Standard-Koordinatensystem von OpenGL

Solange man kein eigenes Koordinatensystem definiert, erwartet OpenGL folgendes Default-System:
(1) Der 0-Punkt liegt in der Mitte der ClientArea.
(2) Die X-Achse geht von `-1.f` (linker Rand) bis `+1.f` (rechter Rand).
(3) Die Y-Achse geht von `-1.f` (unterer Rand) nach oben bis `+1.f` (oberer Rand).

Dieses System hat Vorteile:
(1) Nullpunkt und Achsenrichtungen sind so, wie wir sie aus der Schulmathematik gewohnt sind.
(2) Die Koordinaten sind reelle Zahlen und damit stufenlos zoom- und drehbar.
(3) Es ist unabhängig von der aktuellen Fenstergröße (konstante Breite `2.f` und Höhe `2.f`).

Umrechnen der von Windows (`MM_TEXT`-Mode) gelieferten Mauskoordinaten:
(1) Verschieben des Nullpunkts in die Mitte der `ClientArea`:
Subtrahiere von jedem x die halbe Fensterbreite und von jedem y die halbe Fensterhöhe.
Alle links der Mitte liegenden x und oberhalb der Mitte liegenden y sind jetzt negativ.
(2) Umkehren der Y-Achse:
Multipliziere jedes y mit -1. Alle oberhalb der Mitte liegenden y sind jetzt positiv.
(3) Konvertiere alle x und y vom Datentyp `int` in den Datentyp `float`.
(4) Skalieren auf den Bereich von `-1.f` bis `+1.f`:
Dividiere alle x durch die halbe Fensterbreite und alle y durch die halbe Fensterhöhe.

Useful Links

OpenGL:
www.opengl.org/About/About.html
www.sgi.com/software/opengl
msdn.microsoft.com/library/default.asp?URL=/library/psdk/opengl/openglstart_9uw5.htm
www.xmission.com/~nate/opengl.html
Game development:
cg.cs.tu-berlin.de/~ki/engines.html
www.flipcode.com
www.games-net.de
DirectX:
msdn.microsoft.com/directx