kleiner kommunikation
max@kleiner.com

1 OpenGL in Delphi

OpenGL stellt die derzeit wohl am umfangreichsten dokumentierte 3D-API dar, deren Unterlagen und Funktionen nicht nur online, sondern auch in schnellen Grafikkarten und natürlich gedruckt verfügbar sind. Allerdings ist es unmöglich, in den nächsten Seiten alle Fähigkeiten aufzulisten, jedoch aktuelles zu berichten. Dieser Bericht zeigt den Sprung ins kalte Wasser mit einer prämierten Flugsimulation, um gleich von Anfang an den Power von OpenGL zu zeigen.

1.1 Vom Pixel zur Simulation

In dieser letzten Folge OpenX mit Delphi möchte ich vor allem auch die Bilder sprechen lassen und nur einige Schwerpunkte textlich erwähnen. Mit OpenGL gibt es schon seit geraumer Zeit einige gute Implementierungen von Import Units (Header) welche die beiden grundlegenden DLL's *OpenGL32.dll* und *GLU32.dll* kapseln.

Da sei vor allem die OpenGL 2.0 Übersetzung von www.delphigl.com zu empfehlen, die auch etliche Hilfsfunktionen der ursprünglich von Mike Lischke's *OpenGL12.pas* eingebaut hat und sogar Free Pascal unterstützt. Die Site birgt übrigens auch hervorragende Tutorials zu Delphi.

Wem die Import Unit zuviel Ballast erzeugt, der kann auch kompakten Code auf Basis der reinen GL-API bauen und sogar ohne VCL, da der Compiler nur die nötigsten Funktionen bindet, bspw. in Corecube unter [4] zu finden.

Das fast[1] Open Source - Projekt ging vor etlichen Jahren aus der Bibliothek der Silicon Graphics GL hervor. OpenGL (aktuelle Spezifikation ist 3.1) hat einige hundert Funktionen, die man als Zustandsmaschine verwendet, um 2D und 3D Raster mit einem Z-Bufferalgorithmus durchzuführen. OpenGL bietet dem Entwickler eine Sammlung geometrischer Primitive - Punkte, Zeilen, Polygone, Bilder und Bitübersichten an, die durch eine GLU- und GLUT Bibliothek erweiterbar sind.

Da die Grundfunktionen vor allem Grafikoperationen sind, benötigt man zusätzliche Schnittstellen (siehe Abb. 1) für die Fensterdarstellung der komplexen Grafikobjekte:

- GLX (X-Windows oder Mac)
- GLU (definierte Objekte)
- GLUT (plattformunabhängig)
- Mesa3D (Software-Implementierung des OpenGL Standard)

Diese Bibliotheken dienen zur Erzeugung von Kugeln, Zylindern, Kegeln, Tori (singular Torus;)), NURB-Splineflächen und weiteren Objekttypen (in GLU). Außerdem finden sich dort portable Fenster Funktionen, die häufig verwendete Befehlsfolgen abkürzen (wie GLUT).

Wichtig zu wissen sei die grundlegende Mechanik von OpenGL, die als sequentielle Zustandsmaschine implementiert ist, will heißen: der Zustand einer Grafik-Engine wird über Schaltfunktionen und Parameter eingestellt und bleibt für alle weiteren Funktionsaufrufe erhalten - zumindest solange keine weitere Zustandsänderung eintritt!

Diese Zustandsvariablen spezifizieren Informationen wie Linienbreite, Linienmuster, Farbe, Schattierungsmethode, Nebel oder ganze Gitter. Konkret kann ich über 40 numerische Werte, die als

[1] Grafikkarten Hersteller wie auch Hardware Treiber benötigen eine Lizenz

Argument an `glEnable()` oder `glDisable()` ankommen, z.B. kann ich das virtuelle Wetter beeinflussen indem der Wert `GL_FOG` (welcher den Nebel steuert) zum Einsatz kommt und so den darauffolgenden Zustand ändert.

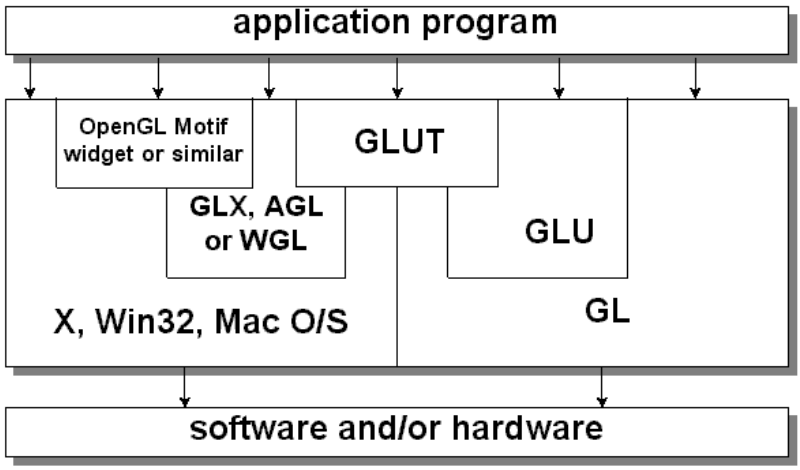
Bspw. sieht ein Würfel mit aktiviertem Licht gezeichnet anders aus als derselbe Würfel ohne Licht. Verändert man einmal einen Zustand, wie z.B. die Farbe, so lassen sich sämtliche Primitive mit dieser Farbe versehen, bis die Farbe erneut wechselt.

Kommen wir zum erwähnten Spiel Air Blast (siehe Abb. 2), daß auf http://www.delphigamer.com/ zu finden ist und anhand dem ich einige Konstrukte vorstelle. Nach der Installation sollte dann die zugehörige Motivation folgen, da sich in der Game-Industrie immer mehr Entwickler aufhalten. Erstaunlich auch, welche von den erwähnten kommerziellen Produkte mit Delphi gebaut sind. Das in Einzelarbeit entwickelte Game, oder eher Simulation, basiert auf GLScene [8] das wiederum eine geniale komponentenorientierte Erweiterung zu Delphi ist und ursprünglich auf Mike Lischke's Arbeit basiert.

http://glscene.sourceforge.net/

GLScene ist eine mächtige Komponentensammlung mit erstaunlich vielen Demos und Tutorials, um z.B. einen Editor für die eigene Engine zu entwickeln oder ganz einfach um die Grafikobjekte eben als Objekte ansprechen zu können! Mit dem zugehörigen Editor und den gekapselten Attributen entwirft man dann Räume, Objekte, Texturen und zugehörige Beleuchtungsmodelle.

Einzig negatives sei die Installation die kompliziert erscheint und auch schon einige gescheitert sind. Sonst wird die Software als Open Source unentgeltlich zum Download jeder interessierten Firma überlassen.



// opengl_arch.tif

Abb. 1: Die Architektur von OpenGL

In OpenGL kommt es immer erst zu einer grundlegenden Initialisierung der Darstellung, dann lassen sich die Features und Zustände aktivieren bzw. deaktivieren und schließlich in den Grundzügen immer wieder neu darstellen. Interessant ist dann der Loop bezüglich dem Neuzeichnen, der in den meisten Fällen mit `onIdle()` oder mit einem Timer die Szene jeweils neu berechnet wie folgendes Schema zeigt.

```
//main event
procedure TGLform.IdleFunc(Sender: TObject; var Done: Boolean);
begin
  //eg: 1000/50 = 20 fps
  openGLControl1.UpdateFrametimeDiff;
  //draw scene
  OpenGLControl1Paint(Self);
  //optimise cpu time
    Sleep(2);
  Done:= false;
  //fps
  if openGLControl1.FrameDiffTimeInMSecs > 2 then
```

```
  glform.HintLabel1.Caption:=  'FPS: '
            +  inttoStr((1000 div openGLControl1.FrameDiffTimeInMSecs));
end;
```

Es gibt auch Alternativen: Man könnte eine eigene Programmschleife aktivieren, welche die Szenen neu zeichnet. In dieser müßte dann aber auch die Botschaften des Betriebssystems einfließen wie das Beispiel CoreCube [4] zeigt.

Eine elegantere Lösung ist meiner Meinung nach das gezeigte OnIdle-Ereignis der Anwendung hierfür zu nutzen. Wenn man darin den Parameter `Done` auf `False` setzt wird dieser Aufruf so oft ausgeführt, wie es nur möglich ist. Das Ereignis wird im Hauptthread der Anwendung aufgerufen, was den positiven Nebeneffekt hat, daß die Nachrichten des Betriebssystems von der VCL verarbeitet werden und man so die VCL in vollem Umfang nutzen kann.

Ein Nachteil der Alternative ist, daß die Anwendung nie zur Ruhe kommt, da die Verarbeitung die Schleife permanent durchläuft. Damit steigt die Prozessorauslastung in der Regel auf 100%. Es sei denn man baut ein `Sleep()` ein, welches ohne Performanceverlust die Last herunterzieht. Die folgende Routine als weiteres Code Schema des sogenannten „Rendern“ merkt vom zusätzlichen `sleep()` nichts und die Frame Rate wird auf wundersame Weise auch nicht beeinflußt:

```
drawScene()
begin
  //set background color
  glClearColor()
  glClear(GL_COLOR_BUFFER_BIT)
  //color of a polygon primitive
  glColor3f()
  //set the polygon
  glBegin(GL_POLYGON)
    glVertex2f()
    glVertex2f()
    .....
  glEnd()
  //display it
  glFlush()
end
```

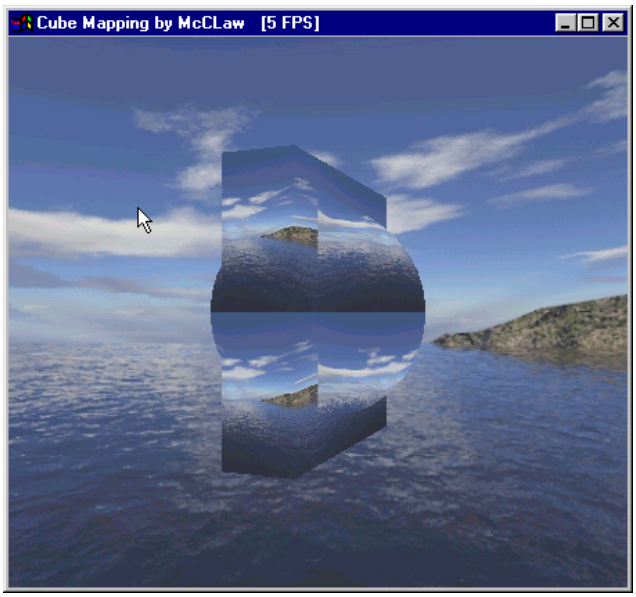
Die sogenannte Framerate, d.h. die Anzahl an darstellbaren Bildern pro Sekunde, sorgt ab ca. 20 fps für eine ruckelfreie Animation. Die Berechnung dazu erfolgt auch gleich im OnIdle Ereignis. Spiele mit einer höheren Framerate werden durch die Trägheit kaum mehr wahrgenommen, es sei denn man suggeriert (manipuliert) mit einem versteckten Bild, z.B. einem Werbetext, den Anwender.

// opengl_airblast.jpg

Abb. 2: Die Simulation zündet

1.2 Play it again

Wem Air Blast zu aggressiv erscheint, dem sei mit CoreCube geholfen. Zur Erzielung von Spiegelungseffekten benutzt man den Modus mit automatischen Reflexionsvektoren (GL 1.3), der wiederum als Zustand aktivierbar ist.



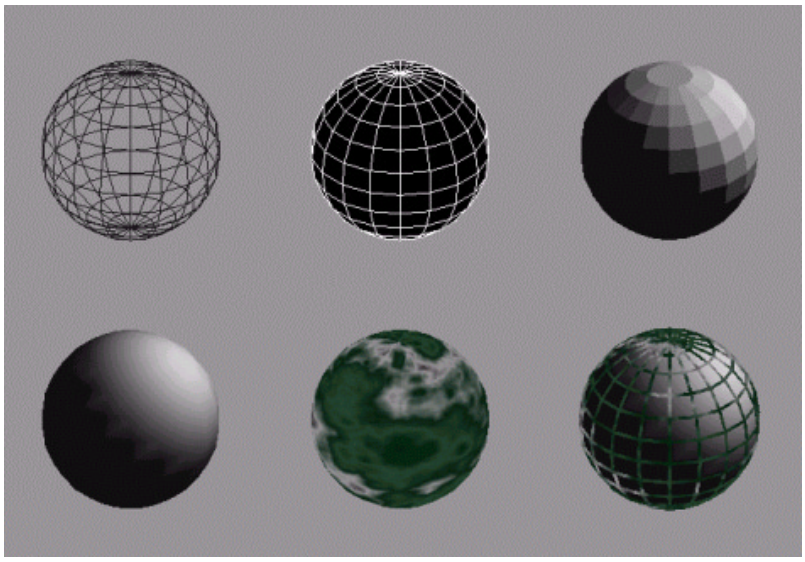
//opengl_cube.tif

Abb. 3: CoreCube dreht sich als Frame

Im CoreCube Beispiel geht es auch um die dynamische Generierung eines Fensters (Petzold und die Win API lassen grüßen) und vor allem um die 3D Darstellung. Die 3D Texturkoordinaten sind mit `GL_REFLECTION_MAP` erzeugt, wobei die betragsmäßig größte Komponente der Koordinaten die verwendete Textur bestimmt. Anbei ein Codemuster, um das „Cubemapping“ zu aktivieren:

```
glEnable(GL_TEXTURE_CUBE_MAP);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
```

Das mit dem „Lighting“ als Beleuchtung ist ein Kapitel für sich, eine Cubemap muß eigentlich nicht 100% exakt sein, es geht plakativ gesagt mehr darum, daß eine Spiegelung auftritt.



//opengl_texture.tif

Abb. 4: Texturen im Aufbau

Texture Mapping ist eine etwas kompliziertere Funktion in OpenGL. Grundsätzlich ist aber jede Grafik als „Überzug“ zu 3D an eine beliebige Fläche anzubinden (zu kleben). OpenGL erwartet die Grafikdaten als eine 24-Bitmap und mit 2er Potenzen Seitenlänge mit quadratischem Mass. Die Funktion erlaubt es auch, eine Textur mit Texturmatrix zu transformieren um dann ein Verschieben, Rotieren, Skalieren zu ermöglichen. Als Beispiel ein Codemuster, um die Textur eine Einheit entlang der X-Achse zu verschieben, und dann 45◦ um die y-Achse zu rotieren (Flugzeug):

```
glMatrixMode(GL_TEXTURE);
glTranslatef(1, 0, 0);
glRotatef(45, 0, 1, 0);
```

Eine weitere Stärke ist die Optimierung der Befehle in Texturen, so kann manZeit und Speicher sparen. Texturdaten sind einmal mit `glTexImage2D` zu transferieren, danach kann man nur noch mit `glBindTexture` referenzieren. Die am häufigsten verwendete Texturen lassen sich im Video-speicher halten (`glAreTexturesResident`, `glPrioritizeTextures`) und komprimieren. Auch das Gruppieren von gleichen Texturen, um Zustandsänderungen zu minimieren, zeigt sich unmittelbar. Dieses Packen hat den Vorteil, daß man nicht jede einzelne Textur im Client aktiviert, sondern das Arbeiten mit Referenzen und Displaylisten vorzieht.

Generell ist OpenGL ganz auf hohe Geschwindigkeit bei der 3D-Darstellung getrimmt. Es beherrscht deshalb kein Raytracing von Haus aus. Für Effekte wie Schatten und Spiegelungen sind die erwähnten Erweiterungen (siehe Abb. 1) einzusetzen, die mit der Zeit auch in Grafikkarten Eingang finden. Erste Treiber für OpenGL 3.0 sind bereits für NVIDIA, ATI uns S3 Chips verfügbar.

Max Kleiner

Literatur und Links:

[1] Deutsche Delphi Site: http://www.delphigl.com/

[2] Rendle, Stritt: Es werde Licht, Der Entwickler, 5 und 6/2002

[3] OpenGL-Projekt: http://www.opengl.org

[4] http://www.softwareschule.ch/download/opengldelphidemo.zip

[5] http://www.softwareschule.ch/download/opengl_delphi_2009.pdf

[6] http://www.delphigamer.com/

[7] http://www.delphi-gems.com/

[8] http://www.glscene.de/