Open Gl - Delphi – Kurs Teil 5: Texturen

Einleitung

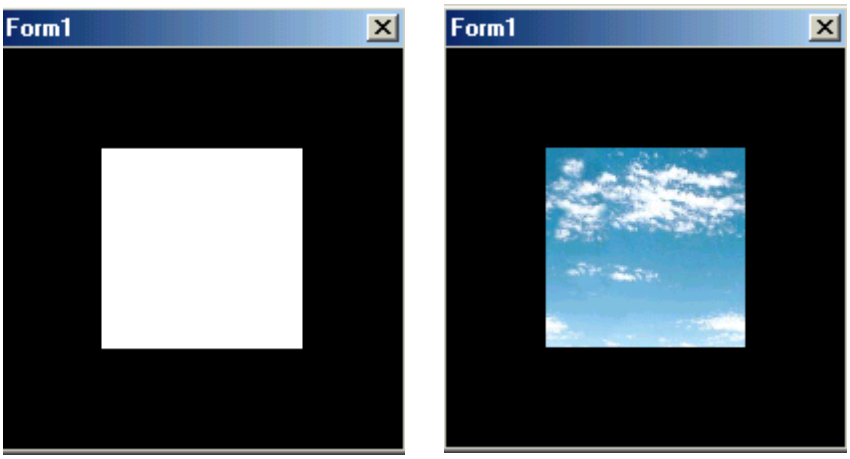
Nun ist es bereits zwei Monate her, dass ich mit diesem Kurs begonnen habe. Nun endlich wollen wir zu einem der wichtigsten Kapitel dieses Kurses kommen: den Texturen! Dieses Kapitel wird zwar nicht ganz so lang, wie das letzte (was aber auch viele Bilder hatte), aber es wird eine Menge wichtigen Stoff vermitteln.

So sei es denn

Lange Rede, kurzer Sinn: bevor wir richtig loslegen müsst ihr euch erst mal was saugen: die Glaux Libary. Diese schon etwas ältere Lib. macht es einem recht einfach, BMP - Bitmaps als Texturen zu laden. Es gibt in Open Gl viele Möglichkeiten Texturen zu laden, aber die Meisten sind mit sehr viel Aufwand verbunden. Daher möchte ich mit euch den recht einfachen weg über die Glaux – Lib gehen. Unten im Download – Part findet ihr diese:
Die dll, die Delphi – Unit zum einbinden und für alle C / C++ - Programmierer die nötigen Header – Files (ß die nennt man doch so, oder?)

Das Ziel

Mittlerehrweile sollte ja an sich jeder von euch dazu in der Lage sein so ein Quadrat wie unten links dargestellt direkt vor unserer Kamera zu erschaffen. Unser Ziel seht ich nun unten rechts:



Also: wir wollen ne nette kleine Himmelstextur auf unseres Viereck kleben (was man auf dem Standbild nicht sieht: der Himmel bewegt sich... wieso, werdet ihr Später erfahren.
Wie macht man das also? Nun: ich gehe davon aus, dass wir das Sample von Kapitel 2 als Grundlage verwenden. Und die Struktur unseres Quadrates sollte so aussehen:

```
glbegin(gl_quads);
glvertex3f(-0.5,-0.5,-3);
glvertex3f(-0.5, 0.5,-3);
glvertex3f( 0.5, 0.5,-3);
glvertex3f( 0.5,-0.5,-3);
glend;
```

Aber was kommt nu? Zunächst müssen wir also unsere Glaux – Unit einfügen. (Bitte achtet darauf, das sich diese im selben Verzeichnis wie euer Projekt und wie die Dll zu befinden hat – die dll darf aber alternativ auch im c:/Windows/System Ordner sein... glaube ich)

```
Uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, opengl,
  ExtCtrls, glaux;
```

Und was dann? Na ja: nun müssen wir erst mal feststellen, das die opengl.pas irgendwie nicht viel taugt. Wieso werdet ihr sagen – bislang lief doch immer alles Perfekt! Das mag auch durchaus sein, aber es fehlen zum Arbeiten mit Texturen zwei wichtige Befehle, die zwar in der „OpenGl32.dll“ vorhanden waren, aber irgendwie nicht ihren Weg in die Open Gl – Unit gefunden haben. Weiß der Teufel warum. Na ja: in jedem Falle fehlen uns diese zwei ziemlich dringend, also holen wir sie uns selber:

```
PROCEDURE glGenTextures(n: GLsizei; VAR textures: GLuint); STDCALL; EXTERNAL
opengl32;
PROCEDURE glBindTexture(target: GLenum; texture: GLuint);  STDCALL; EXTERNAL
opengl32;
```

Falls irgendwer nicht weiß, wohin mit diesem Code, dann verrate ich es euch: ziemlich weit oben im Quellcode direkt hinter

```
implementation

{$R *.DFM}
```

Wer nicht weiß, was diese beiden Befehle bewirken, der soll nicht traurig sein: Ich komme nachher darauf noch mal zurück *teuflisches Grinsen*.
So… nun möchte ich aber den praktischen Teil dieses Teils zurücksetzen und mit euch etwas Theorie machen (tut mir leid: muss so)

Vom UV – Mapping

Also Leute: Nun kommt der Theoretische Teil dieses Teils. Wir wollen uns mit dem UV – Mapping befassen. Vergleichen wir unsere Texturen einfach mal mit einer Tapete und unser Objekt mit einer Wand. Wir wollen nun also unsere Tapete an eine Wand heften. Dabei dürfte folgendes auffallen: es gibt zig Möglichkeiten die Tapete an der wand zu befestigen. Mal längs, mal quer, mal weiter rechts und mal weiter links. In der Realität ist das ganze aber kein Problem: wir wissen, wie wir unsere Tapete anbringen wollen (und können) Aber wie sagen wir Open Gl, wie wir unsere Tapete aufhängen wollen? Nun die Antwort ist einfach: per UV – Mapping! Das ganze läuft nämlich so: jeder 2d Textur hat ja 4 Ecken. Nun weißt Open Gl jeder Ecke eine U und eine V - Koordinate zu. So kommt es, dass die untere linke Ecke jeder Textur die UV – Koordinate 0/0 hat. Die linke obere die Koordinaten 0/1, die rechte obere 1/1 und zuletzt die rechte untere die Koordinaten 0/1 hat. Das gilt es unbedingt zu beachten! Wenn ihr ein Quadrat rendern wollt, dann ist das ja aber auch noch recht einfach. Wollt ihr aber ein Dreieck mit einer Textur versehen, dann werden diese Sachen extrem wichtig, wenn die Textur nicht superschief sitzen soll!
Und wie weisen wir nun einem Punkt eines Quadrates eine UV – Koordinate unserer Textur zu? Na ja: dafür gibt es halt einen recht einfachen Befehl: „gltexcoord2f“ ist unsere Lösung.

Und wie man ach bereits an dem „2f“ sehen kann, verlangt „gltexcoord2f“ zwei Kommazahlen als Parameter. Eine U und eine V – Koordinate. Dieses mit den Kommazahlen ist so: man kann auch krumme Werte eingeben: das ist z.b. bei Dreiecken notwendig: wenn die beiden Seiten unten noch die normalen Koordinaten 0/0 und 0/1 bekommen können, dann sollte die Spitze die Koordinaten 1/0,5 bekommen. Ich hoffe, das ist einleuchtend. Bevor wir nun richtig loslegen möchte ich noch was sagen: Man kann auch eine Textur auf mehrere Objekte verteilen! Wenn ihr z.B. ne recht große Landschaft habt und ihr auf jedes Quadrat der Landschaft dieselbe Textur packt, dann sieht die meistens sehr eintönig und augenfeindlich aus. Wenn ihr die UV – Koordinaten aber so setzt, dass jedes Quadrat nur einen sehr kleinen Teil einer Textur bekommt und so die Textur über die ganze Landschaft verteilt, dann sieht das gleich viel besser aus - glaubt es mir.

Wie laden wir eine Textur?

Also, zunächst benötigen wir eine globale Variable vom Typ „gluInt“. In dieser wird unsere Textur gespeichert. Ich habe diese Variable einfach mal „tex“ genannt (ich sage das blos, damit ihr den Code gleich versteht):

```
Var
  Form1: TForm1;
  tex : gluInt;
```

So... nun müssen wir aber die Textur noch in die Variable laden! Dazu legen wir eine neue Prozedur an. Diese habe mal „InitTextures“ genannt. Darin laden wir nun unsere Textur. Ich poste euch hier mal den Code (Erläuterung kommt gleich):

```
procedure InitTextures;
var
  textur: PTAUX_RGBImageRec;
begin
  textur := auxDIBImageLoadA('himmel.bmp');
  if not Assigned( textur ) then begin
    showmessage(‚Fehler beim Laden der Textur’);
  end;

  glGenTextures(1, tex);
  glBindTexture(GL_TEXTURE_2D, tex);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_linear);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_linear);
  glTexImage2D(GL_TEXTURE_2D, 0, 3, textur^.sizeX, textur^.sizeY, 0, GL_RGB,
GL_UNSIGNED_BYTE, textur^.data);

end;
```

Denn wollen wir mal: Zunächst enthält unsere Prozedur die lokale Variable „Textur“. In dieser Variable speichern wir unseres BMP – Bitmap während dieser Prozedur. Der erste Befehl in der Prozedur ist dann dafür verantwortlich, das unser Bitmap („Himmel.bmp“) geladen wird. Dabei muss die Bitmap natürlich im selber Verzeichnis wie die Anwendung sein, oder der Pfad muss angegeben werden. Die nächsten drei Zeilen sind dafür verantwortlich, dass eine Fehlermeldung ausgegeben wird, wenn die Textur nicht geladen

werden konnte (Falscher Pfad oder Datei einfach nicht vorhanden). Der nächste Parameter generiert dann die richtige Textur. Dazu muss als erster Parameter einfach nur eine Eins angegeben werden (wieso weiß ich selber nicht) und als zweiter kommt der Name für die Textur („tex“ – ihr erinnert euch)
Der nächste Befehl bewirkt nun, das unsere Textur gewählt wird. Das heißt: wir machen Open Gl klar, das wir nun mit „tex“ arbeiten wollen. Dazu müssen wir als Parameter übergeben, um welche Art von Textur es sich handelt (Open Gl kennt auch 3d Texturen) und halt den Namen der Textur mit der wir nun Arbeiten wollen. Auf die nächsten beiden Zeilen möchte ich nun an sich nicht näher eingehen. Hierbei handelt es sich nämlich um die Texturfilter. Texturfilter das sind diese netten Teile, die sich auch bei vielen Spielen bemerkbar machen. Dort darf man dann oft das Trilinearefiltern zuschalten, wenn man genug Rechenpower hat. Wir arbeiten hier mit dem linearen Filtern. Aber diese ganze Sache zu erklären würde den Rahmen dieses Teils sprengen. Eventuell bringe ich später noch einen extra Teil dazu. Kommen wir also zum letzen Befehl. Dieser ist recht schnell erklärt: er weißt „tex“, welches wir ja vorhin angewählt haben die Maße und die Daten von „Textur“ zu. Damit könnten wir ja schon mal ne Textur laden. Wer die Himmelstextur nicht hat, der darf sie sich im Download – Bereich saugen. Ihr könnt aber auch jedes X – Beliebige andere Bitmap nehmen... es muss ja kein Himmel sein. Aber Achtung: Open Gl hat es am liebsten, wenn die Texturen eine Seitenlänge haben, die eine Potenz von zwei sind. (Z.b. 32, 64, 128 oder auch 256.) Das sind halt so die üblichen Texturgrößen.

Packt zusammen, was zusammen gehört

So... an sich haben wir doch nun schon alles: Eine Prozedur, welche die Texturen lädt und eine Methode, mit der wir diese Textur auf unser Quadrat klatschen können. Das erste, was wir nun verändern müssen, ist unsere Form.create – Prozedur, da wir dort zunächst Open Gl mitteilen müssen, dass wir erstens Texturen verwenden wollen und zweitens müssen wir die Prozedur aufrufen, mit der wir unsere Texturen laden. Das sollte dann so aussehen:

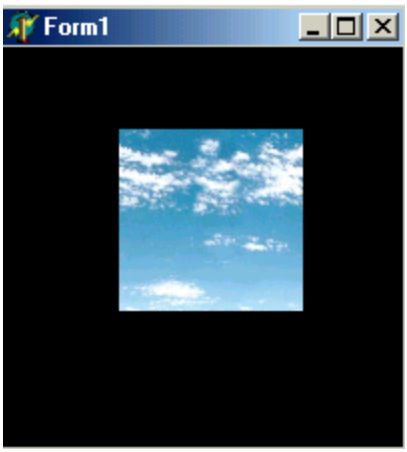
```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.myDC:= GetDC(Handle);
SetupPixelFormat;
myRC:= wglCreateContext(myDC);
wglMakeCurrent(myDC, myRC);
glEnable(GL_DEPTH_TEST);

glenable(gl_texture_2d);  //Texturen aktivieren
InitTextures;             //Unsere Textur laden

glLoadIdentity;
end;
```

Dann hätten wir die Textur schon mal vorbereitet. Nun müssen wir sie nur noch fachgerecht auf unser Viereck kleben. Dazu verändern wir unsere Render – Prozedur:

```
procedure render;
begin
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity;
```

```
glBindTexture(GL_TEXTURE_2D, tex); //Hiermit wählen wir die Textur an
glbegin(gl_quads);
gltexcoord2f(0,0);                 //UV – Zuweisung für den ersten Punkt
glvertex3f(-0.5,-0.5,-3);
gltexcoord2f(0,1);                 //UV – Zuweisung für den zweiten Punkt
glvertex3f(-0.5, 0.5,-3);
gltexcoord2f(1,1);                 //UV – Zuweisung für den dritten Punkt
glvertex3f( 0.5, 0.5,-3);
gltexcoord2f(1,0);                 //UV – Zuweisung für den vierten Punkt
glvertex3f( 0.5,-0.5,-3);
glend;
SwapBuffers(form1.myDC);
end;
```

Nun seid so nett uns startet das ganze... und? Sieht das wie ein Himmel aus? Ja!



Aber wie gesagt: wir wollen unseren Himmel bewegen! Wie das geht, seht ihr gleich....

Die Texturmatrix

Wie bitte? Schon wieder eine Matrix? Igitt! Aber nützt nichts, da müssen wir durch...
Also: für die Texturen gibt es wie für unsere ganze Open Gl – Welt ne extra Matrix. Und was konnte man noch gleich in einer Matrix machen? Ja ganz easy: Verschieben, Skalieren und Rotieren! Und das können wir mit unseren Texturen genau so auch machen: wir brauchen nur per „glmatrixmode(gl_texture);“ in den Texturmodus schalten und schon können wir alles machen, was wir wollen. In meinem Sample habe ich das ganze so gemacht: ich habe eine globale Variable „texmove“, welche ein Integer ist. Diese habe ich bei jedem Render – Vorgang um eins erhöht. Dann habe ich in die Textur – Matrix umgeschaltet, diese zurückgesetzt und dann die Textur per „gltranslate“ bewegt. Allerdings: ein Verschieben um eins würde man nicht bemerken: die Textur sähe genau so aus, wie zuvor. Da sich der Himmel ja recht langsam bewegt, habe ich also beim Verschieben „texmove“ durch 1000 geteilt. Dann bewegt sich der Himmel sehr langsam. Der letzte Schritt war dann nur noch, in die World – Matrix zurück zu schalten, damit wir auch unseres Objekt zeichnen können. Hier ist der neue Code:

```
procedure render;
begin
```

```
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT); //Farb und
Tiefenpuffer löschen
glLoadIdentity;

texmove := texmove+1;                // Bewegung erhöhen
glmatrixmode(gl_texture);            // In den Texturmodus schalten
glLoadIdentity;                      // Die Texturmatrix zurücksetzen
gltranslate(-(texmove/1000),0,0);    // Die nachfolgenden Texturen bewegen
glmatrixmode(gl_modelview);          // Zurück in die World – Matrix

glBindTexture(GL_TEXTURE_2D, tex);
glbegin(gl_quads);
gltexcoord2f(0,0);
glvertex3f(-0.5,-0.5,-3);
gltexcoord2f(0,1);
glvertex3f(-0.5, 0.5,-3);
gltexcoord2f(1,1);
glvertex3f( 0.5, 0.5,-3);
gltexcoord2f(1,0);
glvertex3f( 0.5,-0.5,-3);
glend;
SwapBuffers(form1.myDC);
end;
```

So, damit wären wir auch schon am Ende dieses doch recht wichtigen und auch eindrucksvollen Teiles – endlich Schluss mit den Normalen Vierecken!!! Aber noch sind wir bei weitem mit dem Thema Texturen noch nicht durch: ich werde euch noch beibringen, wie man eine Textur auf mehrere Objekte verteilt (so, dass auf jedem ein Teil des Ganzen ist, was bei großen Landschaften gut sein kann)
Ein weiter Punkt wird die Transparenz und das Blending / Multitexturing, welchem wir nächstes Mal zuleibe rücken werden. Ein letzter Lustiger Part werden auch Texturen, die erst beim Rendern entstehen... das sogenannte „Renderpass“, welches ich mit euch im siebten Teil abhandeln möchte...
Also versauert mir bis dahin nicht und ich wünsche euch fröhliches Coden (und schöne Skyboxen)

Mr_T