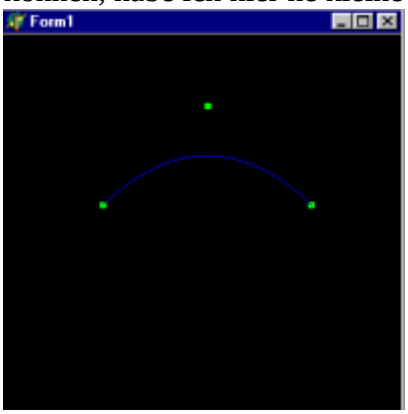
Open Gl - Delphi – Kurs Teil 9: Wie realisiert man Bezier – Kurven

Einleitung

Welcome back to the world of Delphi and Open Glide! Schön euch wiederzusehen! (Bzw. zu lesen)
Ja ja, das Schuljahr neigt sich dem Ende zu, doch soll schon Schluss sein mit lernen? Na ja: in der Schule schon ;-), aber wir hier sind noch lange nicht fertig. (Ich habe soeben mit Frankle die grobe Roadmap für diesen Kurs besprochen... ihr braucht nicht befürchten, dass und vor Teil 15 der Stoff ausgeht :-))
Diesmal wollen wir uns mit den Bezier-Kurven unter Open Gl befassen. Bezier-Kurven? Was ist das? Wozu ist das gut? Kann man das essen?!?
Also vorweg: essen kann man es nicht, aber ansonsten sind diese Teile extrem nützlich... Guckt euch doch mal mit Verstand um... was seht ihr? Euer Zimmer nehme ich an (wer meine Tuts auf der Arbeit ließt, muss völlig von der Rolle sein... oder er ist halt Beamter und genießt damit n recht guten Kündigungsschutz *g*). Und was ist in eurem Zimmer? Na ja, lauter Objekte halt... das Bett, der Computer, mindestens ein Stuhl, wahrscheinlich ein paar Klamotten, die übriggebliebenen Reste der genialen Party gestern... (Dazu fällt mir nun noch n nettes Zitat ein: „Das Leben ist ein verdammt schlechtes Rollenspiel, aber mit einer verdammt guten Grafik“)
An sich sollte uns langsam auffallen, dass hier etwas anders ist, als in euren Open Gl – Programmen... und ich meine damit nicht die Deutlich bessere Beleuchtung und diese geniale Schattenberechnung in Echtzeit ;-). Nein, was ich meine sind die Formen! Guckt euch doch um! Ist denn alles in eurer Welt stur gerade und eckig? Nein, bei mir nicht... gut, es gibt eckiges, aber auch etwas rundes, verzerrtes, usw. so wie zum Beispiel die Rollen meines Schreibtischstuhls oder auch meine Türklinke...
Ok, so weit wollen wir hier noch nicht gehen, aber dieser Teil des Kurses soll dazu gut sein, in eure noch recht eckigen Welten ein paar neue „schwungvollere“ Formen zu bringen.
Genug geschwafelt, auf geht’s!

Bezier-Kurven

Die Frage ist nun, was sind Bezier-Kurven eigentlich?
Um es einfach zu machen: Bezier-Kurven sind an eine Linie, zwischen zwei Punkten, die sich zwischendurch einer dritten annähert. Da die meisten Leute sich darunter kaum was vorstellen können, habe ich hier ne kleine (mit Open Gl erstelle) Grafik für euch:



Es ist sicherlich nun recht einfach zu erkennen, wie man das mit den Bezier-Kurven zu verstehen hat: Der erste und der dritte Punkt (von links nach rechts) sind eindeutig die Endpunkte der Kurve... der mittlere Punkt hingegen wird nur angenähert.
Und soll ich euch was verraten: so ne nette kleine Kurve ist mit Delphi und Open Gl extrem einfach zu realisieren! Ich habe einfach das Sample des 2. Teiles genommen und die Render-Prozedure etwas umgebaut:

```
procedure render;
const
points: array[0..2,0..2] of single = ((-1,0,0), (0,1,0), (1,0,0));
begin
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT); //Farb und
Tiefenpuffer löschen
glLoadIdentity;
glcolor3f(0,0,1);
gltranslate(0,0,-5);
glMap1f(GL_MAP1_VERTEX_3, 0, 1.0, 3, 3, @points[0,0]);
glEnable(GL_MAP1_VERTEX_3);
glMapGrid1f(30, 0.0, 1.0);
glEvalMesh1(GL_LINE, 0, 30);


glcolor3f(0,1,0);
glpointsize(5);
glbegin(gl_points);
glvertex3f(-1,0,0);
glvertex3f(0,1,0);
glvertex3f(1,0,0);
glend;

SwapBuffers(form1.myDC);                    //scene ausgeben
end;
```

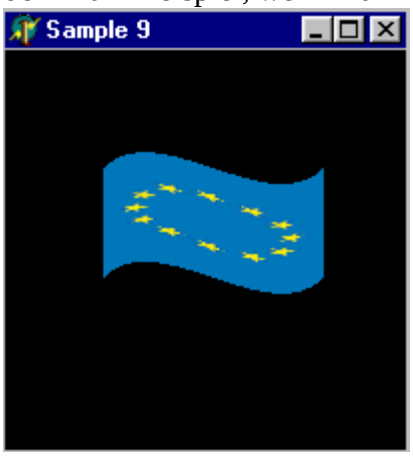
Sieht nun zwar sehr einfach aus, aber ich denke, bei dem einen oder anderen Befehl bedarf es einer Erklärung (und ich meine nicht die Color-Prozeduren *g*):

Das Array von Punkten sollte an sich klar sein: in diesem werden die Beiden Eckpunkte (-1,0,0 bzw. 1,0,0) der Punkt dazwischen (0,1,0) ist der Punkt, den wir annähern.
Der nächste wirklich interessante Befehl ist nun „glmap1f“, welcher ja nun ganze 6 Parameter haben will. Der Befehl selber weißt Open Gl an, intern diese schicke Kurve zu generieren. (Bzw.: Eine Art Kurvengleichung für diese Kurve aufzustellen)
Der erste Parameter gibt an, um was für einen Typ Koordinaten es sich handelt. In diesem Falle sind es dreidimensionale Koordinaten. (Es gibt ja auch Leute, die Open Gl für die 2D Darstellung verwenden)
Der 2. bzw. 3. Parameter stehen für den Bereich, den Open Gl den Enden zuordnet. (ist meistens „0“ und „1“... könnte aber auch was ganz anderes sein)
Der nächste Parameter gibt nun an, wie viele „Zahlen“ bis zur nächsten des selben Typs erfolgen... was damit gemeint ist: in dem Array haben die X-Koordinaten des ersten Punktes und die des Zweiten einen Abstand von genau 3 Zahlen... ich weiß nur nicht, wie ich das anders ausdrücken soll.
Der vorletzte Parameter gibt nun an sich an, wie viele punkte wir übergeben wollen... na klar: 3! Und der letzte ist an sich nun nur noch dazu da, der Prozedur zu sagen, welche punkte sie nehmen soll.. wäre auch sonst etwas sinnlos das ganze.

Der nächste Befehl, der euch unbekannt sein dürfte, ist „glEnable(GL_MAP1_VERTEX_3);“, welcher an sich nur dazu da ist, alles zu aktivieren.
Das Ding ist blos: so sehen wir noch nichts auf dem Schirm... irgendwie auch logo, denn Open Gl hat nun die Formel für ne Kurve und auch den ganzen Kram aktiviert, aber noch nicht wirklich Plan, wie das ganze gezeichnet werden soll... die Sache ist nämlich die: Die normale Grafikausgabe löpt nur über Linien oder Dreiecke... das Ding ist bloß: unsere kurve ist keines von beidem! Der nächste Befehl „glMapGrid1f(30, 0.0, 1.0);“ ändert diese nun aber... er unterteilt unsere kleine Kurve in genau 30 Teile bzw. 30 kurze Linien. Die anderen Beiden parameter stehen an sich nur für den bereich, der unterteilt werden soll... theoretisch könnte man also auch nur die erste hälfte der Linie zeichnen und den Rest entweder gar nicht, oder später...
Der letzte Interessante Befehl ist dann nur noch „glEvalMesh1(GL_LINE, 0, 30);!“ , der den ganzen Kram denn tatsächlich auf dem Schirm bringt. Hier ist auch der Typ angegeben (es ist ja eine Linie) und die darzustellenden Abschnitte... man könnte nun auch erst die erste Hälfte malen, und dann leicht versetzt (durch translaten) die 2. Hälfte – oder man’s gerade braucht...

Bezier-Flächen

Irgendwas stört mich an dem, was wir soeben erschaffen haben... euch auch?
Also am ärgerlichsten finde ich es, dass man mit reinen Linien wenig anfangen kann... was denn zum Beispiel, wenn man ne Flagge zeichnen will? So eine etwa:



Ich weiß, es ist leicht unproportional, aber es geht ja ums Prinzip...
Na ja: da ich ja eine recht umgängliche Person bin und euch auch diese nicht vorenthalten möchte, erkläre ich euch natürlich auch, wie so was geht...

Also, zunächst brauchen wir wieder unsere Punkte. In diesem Falle tue ich so, als handle es sich um ein Rechteck, also bräuchte ich nur zwei Zeilen zur Darstellung. Des weiteren verwende ich zwischen dem jeweils rechten und Linken Eckpunkt jeweils zwei Kontrollpunkte zwecks Krümmung... mein Array sieht daher so aus:

```
const
points : array [0..1, 0..3, 0..2] of glfloat = (((-2,1,0),(-1,2,0),(1,0,0),(2,1,0)), ((-2,-1,0), (-1,0,0), (1,-2,0), (2,-1,0)));
```

Das mag nun zwar etwas merkwürdig aussehen, aber an sich ist es total einfach. Die ersten vier punkte stehen für die obere Reihe und die Punkte 5-8 für die untere Reihe... die Punkte 2 und 3 (bzw. 6 und 7) sind jeweils die, die angenähert werden.

Nun müssen wir wie vorhin erst mal Open Gl dazu anweisen, intern ne Art Gleichung für das ganze aufzustellen. Das machen wir diesmal so: (nicht erschrecken!)

```
glMap2f(GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4, 0.0, 1.0, 12, 2, @Points[0,0,0]);
```

Nanu? Wozu denn so viele Parameter? Na ja es ist an sich recht einfach zu erklären: Open Gl betrachtet die Anzahl der Spalten und reihen getrennt... das heißt, dass sowohl die Reihen, als auch die Spalten in einem Bereich von Null bis Eins dargestellt werden. (Deshalb das zweimalige Aufkreuzen der Parameter „0.0“ und „1.0“)
Der 4. Parameter dürfte bekannt sein: er stellt dar, wie viele Werte es von X-Wert zu X-Wert sind (siehe einfache Kurve...)
Der 5. Parameter ist auch klar: hier geht es drum, dass pro Zeile 4 Koordinaten liegen.
Die beiden vorletzten Parameter dürften nun aber nicht ganz klar sein:
Die „12“ bedeutet, dass es von Anfang einer Zeile bis zum Anfang der nächsten 12 Werte sind (Anzahl der Punkte pro Zeile * Anzahl der Koordinaten pro Punkt)... der vorletzte Parameter bedeutet nun, dass wir insgesamt 2 Zeilen bzw. Spalten zeichnen wollen.
So, nun müssen wir natürlich auch noch dieses ganze wieder wie vorhin in Teile einteilen. Bei Flächen verwenden wir hier diesen Befehl:

```
glMapGrid2f(50, 0.0, 1.0, 50, 0.0, 1.0);
```

Wunder oh Wunder: alle Parameter sind wie eben in doppelter Anzahl gegenüber der Linienvariante vorhanden. So kann man nun sowohl für die Spalten, als auch für die Zeilen angeben, mit welcher Genauigkeit gerendert werden soll und welcher Abschnitt es sein darf... hier war ich mit der Genauigkeit mal großzügig ;-)

Nun muss der ganze Kram zum guten Abschluss aber noch auf den Schirm gebracht werden...

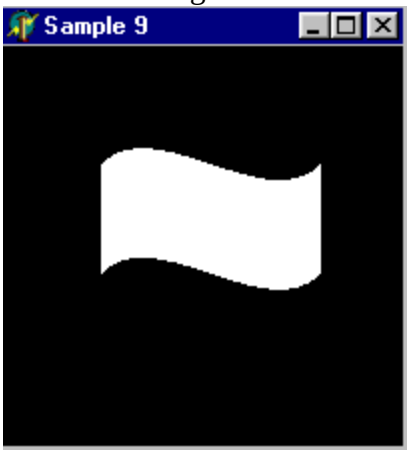
```
gltranslate(0,0,-10);
glEvalMesh2(GL_FILL, 0, 50, 0, 50);
```

Irgendwie nicht wirklich erschreckend, oder?
Aber haben wir nicht noch was vergessen? Ja, genau, wir haben ja noch gar nicht Open Gl gesagt, dass es so was überhaupt kann... (besser: wir heben ein „glEnable(); vergessen)
In diesem Falle habe ich die nötige Procedure „glEnable(GL_MAP2_VERTEX_3);“ aber einfach an die On-Create – Prozedur des Formulars angehängt... es muss ja nun wirklich nicht immer wieder aktiviert werden... einmal reicht doch...
Der ganze Render-Code sieht bei mir nun bislang so aus:

```
procedure render;
const
points : array [0..1, 0..3, 0..2] of glfloat = (((-2,1,0),(-1,2,0),(1,0,0),(2,1,0)), ((-2,-1,0), (-1,0,0), (1,-2,0), (2,-1,0)));
begin
glMap2f(GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4, 0.0, 1.0, 12, 2, @Points[0,0,0]);
glMapGrid2f(50, 0.0, 1.0, 50, 0.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity;
gltranslate(0,0,-10);
glEvalMesh2(GL_FILL, 0, 50, 0, 50);

SwapBuffers(form1.myDC);
end;
```

Wenn wir das ganze nun Starten, ist das Ergebnis aber irgendwie noch nicht ganz so schön...



Man sieht sofort, was fehlt: Die Textur! Und darum kümmern wir uns nun...

Bezier-Kurven mit Textur

Bevor wir uns richtig um Texturen kümmern, will ich euch erst mal ne neue Library zum Laden von Texturen vorstellen: die „glBMP“ – Lib. Man kann sich diese entweder unter „http://www.cfxweb.net/~delphigl/projects/glbmp.htm“ oder von unserer Site runterladen.
Aber was ist nun so toll an dieser Lib?
Na ja: die Vorteile gegenüber der Glaux-Library, die wir bislang gebraucht haben, liegen zumindest für mich auf der Hand:
Sie ist deutlich kleiner, leichter zu bedienen und man kann mit ihr auch andere Vormate außer BMP, wie zum Beispiel PNG oder auch JPG Dateien öffnen.

Wie man die Unit bedient ist auch ganz einfach:
Zunächst muss man bei Uses halt „glbmp“ eintragen.
Dann müssen wir unsere Variable für die Textur noch ändern:
Bisher sah die ja noch so aus: „textur : gluint;“ ...
Nun muss das ganze aber so aussehen:
„textur : Tglbmp;“

Soll ich euch was verraten? Ja? Ok: das Laden einer Textur können wir nämlich nun mit nur 3 Zeilen! Um eine Textur zu laden verwenden wir ab nun diesen Code:

```
procedure inittextures;
begin
textur := tglbmp.Create;
textur.LoadImage('Bild.bmp');
textur.GenTexture;
end;
```

Das ist alles...
Ach ja: evtl. sollte ich euch nun noch zeigen, wie man nun diese Textur „bindet“:
„textur.bind;”

Das war alles!
Und wieso habe ich euch diese einfache Methode nicht schon viel früher gezeigt? Ganz einfach: Ich wusste es nicht besser :-(Na ja: Zurück zum Thema...

Zurück zum Thema

Ämm... ja...
Wir wollten also unsere Flagge mit einer Textur versehen. Wie man die Textur lädt, das wissen wir ja nun, aber irgendwie müssen wir ja auch noch die Texturkoordinaten angeben, was bei so einem gekrümmten Objekt nicht so einfach scheint :-/
Aber auch da haben die netten Open Gl – Entwickler an uns gedacht :-)
Und zwar: Zunächst müssen wir alle Textur-Koordinaten ebenfalls wie die Vertexe inj ein Array schreiben. Bei mir sieht das Array so aus:

```
coords : Array[0..1, 0..1, 0..1] of GlFloat =(((0.0, 0.0),(0.0, 1.0)), ((1.0, 0.0),(1.0, 1.0)));
```

Wie ihr bereits wisst, brauchen wir für jeden Eckpunkt zwei Textur - Koordinaten. Dazu kommt, dass wir für beide Dimensionen unseres Curved - Surfaces diese haben müssen. Daher die Struktur des Arrays...
Nun müssen wir die Textur-Koordinaten ja noch zuweisen. Dies geschieht mit dem folgenden Befehl:

```
glMap2f(GL_MAP2_TEXTURE_COORD_2, 0.0, 1.0, 2, 2, 0.0, 1.0, 4, 2, @Coords[0,0,0]);
```

Kommt euch das nicht bekannt vor? Also mir schon: Es ist derselbe, mit dem man auch Open Gl dazu angewiesen hat, quasi ne Gleichung für das Curved Surface aufzustellen. In diesem Falle weißt man dem Befehl zu, dass er hat die Textur-Koordinaten anhand unseres Arrays berechnen soll... alle anderen Parameter sind Erklärungstechnisch mit der Beschreibung von vorhin identisch.
Nun müssen wir an sich nur noch mit Hilfe von

```
glEnable(GL_MAP2_TEXTURE_COORD_2)
```

bei der Initialisierung das Berechnen der Koordinaten aktivieren und wir sind an sich fertig.

Um euch noch mal einen Überblick zu geben zeige ich noch mal, wie nun meine Render-Prozedur aussieht:

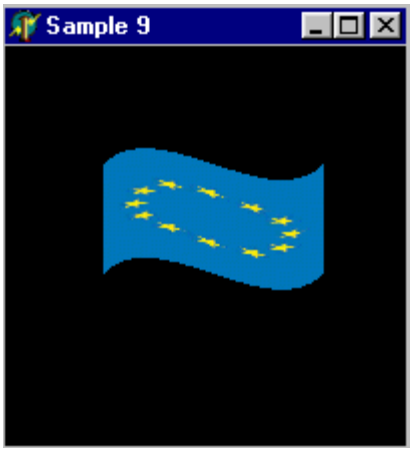
```
procedure render;
const
points : array [0..1, 0..3, 0..2] of glfloat = (((-2,1,0),(-1,2,0),(1,0,0),(2,1,0)), ((-2,-1,0), (-1,0,0), (1,-2,0), (2,-1,0)));
coords : Array[0..1, 0..1, 0..1] of GlFloat =(((0.0, 0.0),(0.0, 1.0)), ((1.0, 0.0),(1.0, 1.0)));
begin

glMap2f(GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4, 0.0, 1.0, 12, 2, @Points[0,0,0]);
textur.bind;
glMap2f(GL_MAP2_TEXTURE_COORD_2, 0.0, 1.0, 2, 2, 0.0, 1.0, 4, 2, @Coords[0,0,0]);
glMapGrid2f(50, 0.0, 1.0, 50, 0.0, 1.0);

glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT); //Farb und Tiefenpuffer löschen
glLoadIdentity;                          // World Matrix zurücksetzen
gltranslate(0,0,-10);
glEvalMesh2(GL_FILL, 0, 50, 0, 50);

SwapBuffers(form1.myDC);                 //scene ausgeben
end;
```

So, das war’s! Wir brauchen nicht mehr und nicht weniger um dieses zu zeichnen:



Ich hoffe, es hat euch einigermaßen Spaß gemacht und war nicht zu schwer zu verstehen.
Und mit diesen Worten hoffe ich, dass ihr auch beim nächsten Teil wieder reinguckt, wenn es ums Thema „Selection, oder wie Progranmiere ich einen Lichtschalter“ geht.

Mr_T