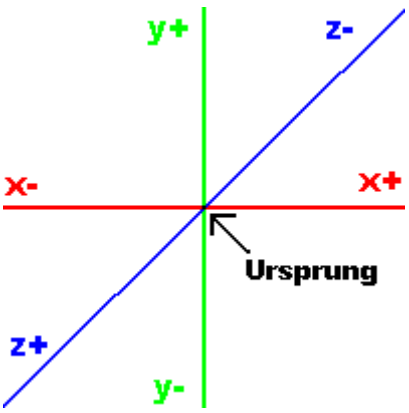
Open Gl - Delphi – Kurs Teil 3: Objekte und die World Matrix

Einleitung

Mit den Fingern Knack so... nun ist es also schon wieder soweit: der dritte Teil meines Delphi – Open Gl – Kurses steht an. Diesmal (Mitlehrerweile schon 4 Wochen nach dem ersten Teil) wollen wir endlich mal was zeichnen! Ich hoffe, dieser Teil wird nicht ganz so Theorielastig, wie der Letzte. Aber ich kann euch beruhigen: wahrscheinlich wird dieser Teil ein großer Spaß... auch wenn ich leider doch mit ein wenig Theorie anfangen muss, aber wenigstens dürfte diesmal am ende wenigstens was zu sehen sein (wenn man das Sample des letzen Teiles startet, hat man noch einen schwarzen Bildschirm). Also los!

Die Open Gl Basics

So Leute nun wird es Zeit, euch über die Basics von Open Gl aufzuklären. Also wie gesagt: Wenn ihr bislang eure Szene startet, so werdet ihr bemerken, dass es sich nur um einen schwarzen Bildschirm handelt. In diesen wollen wir nun was „reinrendern“. Bloß wohin genau? Gibt es irgendwelche Koordinaten?
Ja, es gibt welche! Es handelt sich dabei um die Koordinaten der World Matrix oder auch Welt Koordinaten. Viele von euch haben ja schon mal in Mathe ein Koordinatensystem gesehen, und unseres ist diesem nicht unähnlich. In der Mitte des Ganzen ist der Ursprung, also der Punkt, bei welchem alle Koordinaten den wert null haben. Neben der X – Achse, welche links negativ ist und rechts positiv, und der Y – Achse, die unten negativ und oben positiv ist, gibt es in Open Gl auch noch die Z – Achse. Die verläuft aber nicht, wie viele denken, von hinten positiv nach vorne negativ, sondern sie wird nach hinten hin negativ!!! Dies ist insbesondere für Direkt X – Freaks sehr schwer am Anfang, da es dort genau anders herum ist. Man sollte also darauf aufpassen, wenn man sich keine riesigen Fehler einfangen will. Ich habe dazu mal eine Kleine Grafik gemalt:



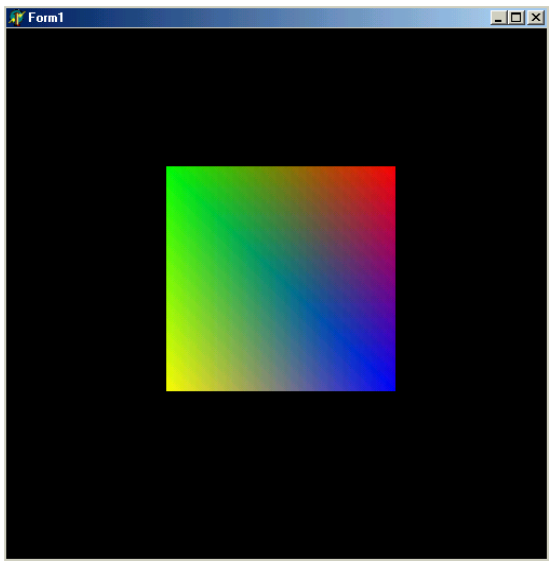
So... das hätten wir schon mal. Aber wo genau sind wir nun? Tja, das ist leicht beantwortet: wir stehen dort, wo alles begann – auf dem Ursprung. Und was können wir alles sehen? Das hatten wir im letzen Kapitel doch schon mal: alles, was nicht weiter als 100 Einheiten von uns weg ist und nicht gerade außerhalb unseres 45° Winkels liegt... also eine Ganze Menge. Aber wir müssen eines Beachten: dank unserer Einstellungen darf das Objekt nicht näher als 1 sein, sonst sehen wir gar nichts!!!

Ich will was Zeichnen!!!

Ja ja kommt ja schon...
Also, wie malt man denn nun überhaupt was?
Auch da gibt es 2 wichtige Basics:

1) Alle Objekte werden zwischen glbegin( ) und glend; gerendert. In der Klammer von glbegin gibt man dazu den Objekttyp an. Z.B. ob es sich um Punkte, Linien, Dreiecke oder Quadrate handeln soll.
2) Alle Objekte werden durch Punkte gezeichnet. Der befehl für einen Punkt heißt glvertex. Glvertex kann aber verschieden viele Parameter haben. So gibt es eine Variante glvertex2f und eine glvertex3f. Der Unterschied ist klar: glvertex2f verlangt 2 Floats als Parameter (x und y Koordinate)... glvertex3f hingegen 3 (x, y und z Koordianten)

Und was sagt uns das? Tja... man kann mit Open Gl nicht nur 3d, sondern auch 2d zeichnen, was aber nicht unbedingt Sinn der Sache ist. Ich habe mir überlegt, dass wir folgende Szene erreichen wollen:



Was brauchen wir denn zunächst dazu? An sich benötigen wir zunächst mal ein Quadrat. Und wie zeichnen wir das? Wie oben bereits gesagt mit 4 Punkten. Wir haben ja bereits unsere Render – Prozedur. In dieser habe ich beabsichtigt eine keine Lücke gelassen. In diese kommt nun unserer Code. Wir wollen mal sehen, was passiert, wenn wird diesen dort eintippen:

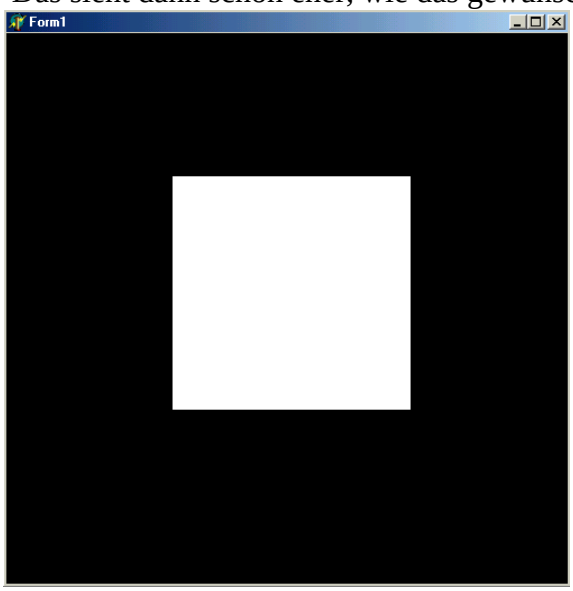
```
Glbegin(gl_quads);        //wir wollen ein Viereck zeichnen
  glVertex3f( 1 , 1, -2);
  glVertex3f( -1 , 1, -2);
  glVertex3f( -1 , -1, -2);
  glVertex3f( 1 , -1, -2);
glend;
```

So, das will ich erst mal erläutern:
Also mit glbegin(gl_quads); beginnen wir, Vierecke zu zeichnen. Danach folgen die Vier Eckpunkte und zuletzt das Abschleißende glend. Aber was ist das, wenn wir das nun starten? Die Szene ist urplötzlich ganz weiß!!! Wie kann das denn nun? Die Erklärung für dieses Phänomen ist sehr einfach: Der nächste Punkt, den wir sehen können, hat die Koordinaten (0,0,-1) das heißt: wir sind sehr nahe drauf auf unserer Kamera! Unser Objekt ist schlich zu

nahe, um ins Bild zu passen! Was tun wir folglich dagegen? Wir zeichnen es etwas weiter weg. Mit diesem Code zeichnet man es z.b. schon mal 4 Einheiten weiter nach hinten, als bisher. Das hilft schon sehr.

```
Glbegin(gl_quads);        //wir wollen ein Viereck zeichnen
  glVertex3f( 1 , 1, -6);
  glVertex3f( -1 , 1, -6);
  glVertex3f( -1 , -1, -6);
  glVertex3f( 1 , -1, -6);
glend;
```

Das sieht dann schon eher, wie das gewünschte Viereck aus:



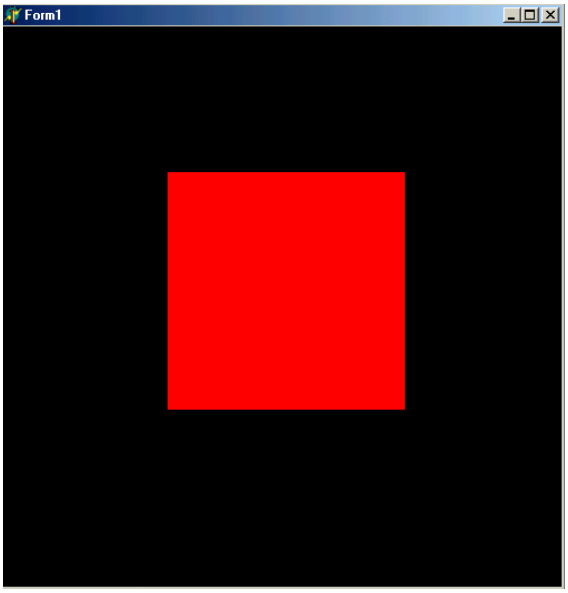
Aber irgendwie sieht das noch sehr eintönig aus. Aber das ändern wir nun.

Ich sehe... Farben ;-)

So... wie bekommen wir nun also etwas Farbe in unseres Viereck? Also, falls nun schon irgendwer auf den Gedanken gekommen ist, jedes Pixel für Pixel einfach nachzuzeichnen, wird er gelyncht! Das geht viel einfacher! Man ist es ja bereits gewohnt, das sich Farbe in PCs aus rot, grün und blau zusammensetzt. Und so läuft das auch hier! Mit dem Befehl glcolor3f, welcher drei Parameter verlangt (sieht an ja auch an dem „3f“) kann man nun eine Farbe setzen. Aber es gibt eines zu beachten: Die Farbwerte werden zwischen 0 und 1, also als Float angegeben! Wer nun also gerne seine HTML – Farben verwenden wollte, der sollte alle Farbwerte schleunigst durch 256 teilen, damit das auch hinhaut! Also: wir schreiben nun vor dem Zeichnen unseres Vierecks ein glcolor3f, mit dem wir eine rote Farbe erzeugen wollen:

```
Glcolor3f(1,0,0)          //rote Farbe
Glbegin(gl_quads);        //wir wollen ein Viereck zeichnen
  glVertex3f( 1 , 1, -6);
  glVertex3f( -1 , 1, -6);
  glVertex3f( -1 , -1, -6);
  glVertex3f( 1 , -1, -6);
glend;
```

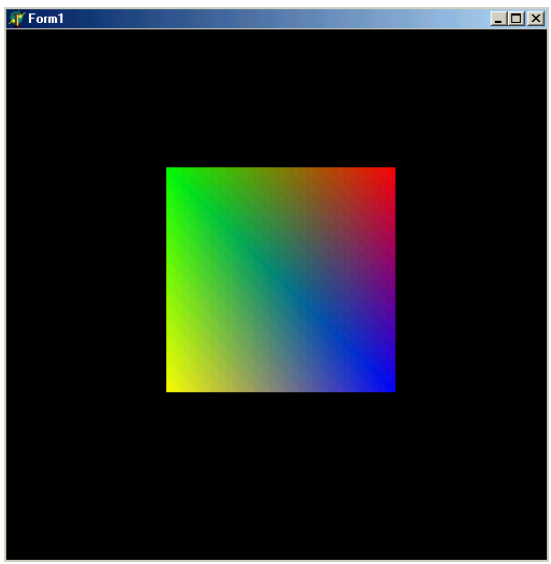
Dann sieht das Ganze schon mal so aus:



Aber wo bekommen wir nun diese schönen Farbverläufe her? Das ist an sich ganz einfach: Man kann für jeden Punkt eine einzelne Farbe angeben. Da ja alle punkte dank unseres glbegin(gl_quads) zu einem Viereck verbunden wird, werden die Farbverläufe einfach automatisch berechnet! Dies kann manchmal sehr nützlich sein. Daher brauchen wir unseren Quadratcode nur so umschreiben:

```
Glbegin(gl_quads);        //wir wollen ein Viereck zeichnen
  glcolor3f(1,0,0)          //rote Farbe
  glVertex3f( 1 , 1, -6);
  glcolor3f(0,1,0)          //grüne Farbe
  glVertex3f( -1 , 1, -6);
  glcolor3f(1,1,0)          //gelbe Farbe
  glVertex3f( -1 , -1, -6);
  glcolor3f(0,0,1)          //blaue Farbe
  glVertex3f( 1 , -1, -6);
glend;
```

Und schon haben wir unseren Farbverlauf!



Vom Schieben, Drehen und Skalieren

An sich ist Open Gl ja richtig brutal und kriminell!
Hier werden einige Autos und andere Objekte durch die Gegend geschoben, woanders werden Dinge komplett verdreht und irgendwo skaliert man aus einer Mücke einen Elefanten! Einfach kriminell!
Aber so schlimm ist es nun auch wieder nicht. Was wäre das für eine Welt, ohne *Animationen*? Wäre nicht gerade lustig, würde ich sagen... Na ja, was soll also das Geschwafel? Ich möchte euch nun beibringen, wie das eben genannte also funktioniert.
Ich fange mit dem Verschieben an:

Zum Verschieben reicht in Open Gl ein einziger Befehl! Nämlich gltranslate. Dieser Befehl verlangt - wie sollte es auch anders - 3 Parameter. Diese sagen ganz easy aus, wohin verschoben werden soll. Wenn ich also vor unserem Viereck „gltranslate(0,1,-2)“ aufrufe, dann verschieben wir es um ca. zwei Weltkoordinaten nach hinten und um eine nach oben. Das heißt also auch, dass wir unser viel zu nahes Anfangsquadrat hätten lassen können und einfach „gltranslate(0,0,-4)“ davor hätten aufrufen können, womit ich euch aber erst nicht belasten wollte. Nun werde ich erst mal zum drehen kommen (keine Sorge, ich mache noch ein Sample am Ende mit allen befehlen zusammen... versprochen)

Es dreht sich

Das Drehen geht an sich genauso einfach von der Hand, wie das Verschieben. Zum Drehen braucht man nämlich nur den Befehl „glrotate“. Dieser Befehl verlangt insgesamt 4 Parameter. Der erste ist der Winkel in Grad. Die andren Drei stehen wieder für eine X,Y und Z – Koordinate, welche die Achse beschrieben soll, um die rotiert wird. Wenn man also „glrotate(45, 0, 1, 0)“ aufruft, dann dreht sich alles nachfolgende um 45° um die Y – Achse. Was man nun zuerst tut (Drehen oder Verschieben) ist jedem selber überlassen, aber es hat sehr verschiedene Auswirkungen! Das liegt daher, das immer um den Punkt gedreht wird, um den eben „translatet“ wurde! Spielt doch einfach etwas damit (und mit den Parametern) rum. Ich werdet den Bogen schnell raushaben!

Na denn: Skalieren

So, womit wir auch schon beim skalieren wären. Lasst es uns kurz machen: zum Skalieren braucht man nur den Befehl „glscale“, der insgesamt drei Parameterchen haben möchte. Die erste gibt die Skalierung der X – Richtung an, womit auch klar wäre, das die beiden anderen für Y und Z zuständig sind. Wenn man nun also ein Objekt gleichmäßig vergrößern oder verkleinern will, so muss man einfach alle Werte gleichmäßig erhöhen oder erniedrigen. Eine Sache muss aber noch beachtet werden: es handelt sich bei den werten um Multiplikatoren und nicht um Summanden!!! Einige Leute denken: „Wenn du null als Parameter angibt, dann verändert sich Größe null“ und wundern sich dann, wenn urplötzlich nichts mehr zu sehen ist. Also: wenn ihr die Größe eine Koordinate nicht ändern wollt, dann solltet ihr 1 angeben, wollt ihr sie verdoppelt, so gebet 2 an und wollt ihr sie halbieren, so hat der Parameter 0.5 zu heißen! Ach ja noch was: Ein Objekt wird immer vom Ursprung aus skaliert. Damit macht es dann doch einen Unterschied, ob wir ein Objekt gleich an die richtige stelle schieben, oder aber sie am Ursprung erstellt und dann verschiebt.....

Und nun: Das Sample

So, nun möchte ich also mit euch ein kleines Sample bauen.
Das ganze soll so aussehen:
Wir verwenden unseres Quadrat von vorhin und versuchen, dieses

1) von der Größe her pulsieren zu lassen
2) langsam immer von oben nach unten zu bewegen
3) es ständig um die eigene Z – Achse rotieren so lassen

Damit wäre unser Auftrag klar. Zunächst benötigen wir insgesamt fünf neue globale Variablen:

```
var
  Form1: TForm1;
  scale : single = 1;          //Für die Skalierung
  rotation : integer = 0;      //Zahl in °, um die gedreht wird
  move : single = 0;           //Größe der Bewegung
  scaledir : boolean;          //Richtung der Skalierung (größer / kleiner)
  movedir : boolean;           //Richtung der Bewegung (rauf / runter)
```

So... damit hätten wir schon mal die wichtigsten Variablen beisammen. Nun müssen wir diesen nur noch werte zuweisen. Dazu habe ich mir gedacht, nehmen wir uns einen neuen Timer, den wir auf etwa 50 stellen. Dort müssen wir dann die werte zuweisen:

```
procedure TForm1.Timer2Timer(Sender: TObject);
begin
if scale > 1.2 then //ist die höchste stufe der Skalierung erreicht?
begin
scaledir := false;  //Ja : Skalierung umkehren
end;

if scale < 0.8 then //ist die niedrigste Stufe der Skalierung erreicht?
begin
scaledir := true;   //Ja : Skalierung umkehren
end;

if scaledir = true then   //Soll die Skalierung größer werden?
begin
scale := scale + 0.02;    //JA: Scale erhöhen
end
else
begin
scale := scale - 0.02;    //Nein: Scale ernidrigen
end;

inc(rotation);    //Die rotation in "°" erhöhen

if move > 1 then //ist der höchste Punkt der Bewegung erreicht?
begin
movedir := false;  //Ja : Bewegung umkehren
end;
```

```
if move < -1 then //ist der niedrigste Punkt der Bewegung erreicht?
begin
movedir := true;   //Ja : Bewegung umkehren
end;

if movedir = true then   //Soll die Bewegung nach oben gehn?
begin
move := move + 0.02;    //JA: Move erhöhen
end
else
begin
move := move - 0.02;    //Nein: Move ernidrigen
end;

end;
```

Ich denke mal, dass ich diesen Quelltext nicht unbedingt näher erklären muss. Er erklärt sich ja praktisch von selbst: zunächst wird die Skalierung, dann die Rotation und zuletzt die Bewegung bearbeitet. Wenn man das ganze nun aber startet, dann sieht man nichts! Besser: es bewegt sich nichts! Das ist aber auch nicht schwer zu erraten, wieso. Ganz einfach: wir verarbeiten die neuen Werte noch nicht. Dazu müssen wir noch unsere Render – Prozedur noch etwas abändern:

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


gltranslate(0,move,0);     //wir bewegen alles etwas nach oben oder unten
glscale(scale,scale,1);    //Nun skalieren wir die Szene
glrotate(rotation,0,0,1);  //Und ab in die Drehung um die Z-Achse ;-)


glBegin(GL_Quads);
  glcolor3f(1,0,0);
  glVertex3f( 1 , 1, -6);
  glcolor3f(0,1,0);
  glVertex3f( -1 , 1, -6);
  glcolor3f(1,1,0);
  glVertex3f( -1 , -1, -6);
  glcolor3f(0,0,1);
  glVertex3f( 1 , -1, -6);
glEnd();



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

Das wäre auch schon alles! Wenn wir nun das ganze starten dürften wir ein quietschbuntes Viereck sehen, welches gegen jedes Gesetz der Physik verstößt. Aber das ist an sich ja halb so

schlimm: es war ja quasi Sinn der Sache, euch mal irgendwie alles zusammen zu zeigen. Wer dort noch nicht ganz durchgestiegen ist: saugt euch das Sample und spielt etwas mit den Parametern rum: ihr werdet dabei ne Menge lernen!

Nachwort

So, das war's auch „schon" mit Teil 3 meines Open Gl Kurses. Ich hoffe, diese Acht Seiten waren etwas Unterhaltsamer, als die sieben Seiten aus dem 2. Teil und ich hoffe, dass ich euch in zwei Wochen wieder begrüßen darf, wenn wir zum Thema „Zeichenmethoden" kommen. Dort werde ich euch beibringen, wie man Punkte, Linien, Dreiecke, Rechtecke und Polygone in den verschiedensten Zeichenstärken, usw. Macht. Zudem machen wir noch einen kleinen Abstecher in die Welt der 3d – Objekte (in der wir ab dann auch bleiben werden). Ich werde euch also zeigen, wie man Pyramiden, Würfel und Kugeln auf den Screen bekommt. Ich hoffe, wir sehen uns wieder.

Mr_T