Open Gl - Delphi – Kurs Teil 4: Im Zeichen des Zeichenstiftes

Einleitung

Wie bitte? Sind die 2 Wochen schon wieder um? Die kommen mir momentan so gummiartig vor... werden immer kürzer...
Aber was will man machen... da müssen wir nun durch... (ist die Überschrift nicht genial?) Na ja: wir wollen uns heute mit Open Gl – Objekten beschäftigen. Also, denn (ich bin kein Mensch großer Vorworte... dafür brabble ich danach um so mehr) lasst und loslegen.

Tausend und ein Parameter

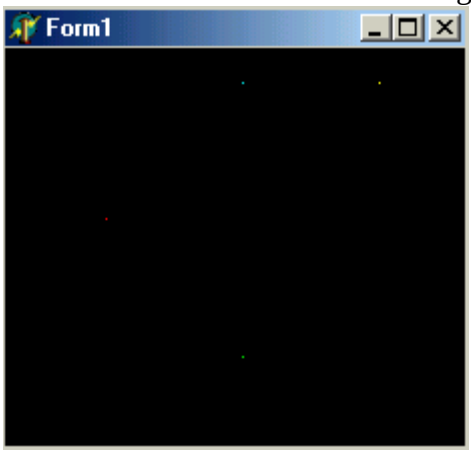
Na ja: es sind nicht ganz so viele, ab doch schon eine beachtliche Anzahl. Ich spreche hiermit von dem einzigen Parameter von Glbegin(), welcher an sich alles entscheidet. Dieser eine Parameter sorgt dafür, das eine Szene überhaupt so aussieht, wie sie aussieht und nicht anders (ß Irgendwie war das überflüssig :-P) Wir wissen ja seit dem letzen Kapitel alle, das an sich die ganze Szene nur durch glvertex3f bestimmt wird. Dieser Befehl legt ja wie bekannt Punkte fest. Der Parameter in glbegin() gibt nun an sich nur an, wie diese Punkte miteinander verbunden werden sollen. Also, lets go!

Punkte und Linien

So... wir beginnen also mit dem einfachsten: den Punkten!
Wenn man als Parameter „GL_POINTS“ angibt, dann werden alle angegebenen Punkt auch wirklich nur als einzelne Punkte gezeichnet. Also an sich so, wie man sie angibt. Ein Beispiel:

```
glbegin(gl_points);
glcolor3f(1,0,0);
glvertex3f(-0.5,0,-2);
glcolor3f(0,1,0);
glvertex3f(0,-0.5,-2);
glcolor3f(1,1,0);
glvertex3f(0.5,0.5,-2);
glcolor3f(0,1,1);
glvertex3f(0,0.5,-2);
glend;
```

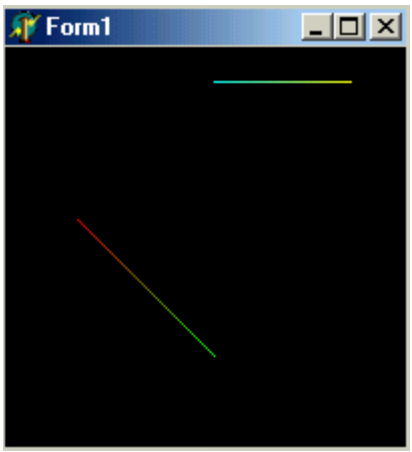
Das dürfte dann etwa so ein Bild ergeben:



Man sieht: Drei bunte, einsame Punkte. Aber das ändern wir nun: wir malen Linien!
Um das ganze zu Linien umzubauen braucht man nur den Parameter „gl_lines“ angeben!
Der Quelltext hieße dann so:

```
glbegin(gl_lines);
glcolor3f(1,0,0);
glvertex3f(-0.5,0,-2);
glcolor3f(0,1,0);
glvertex3f(0,-0.5,-2);
glcolor3f(1,1,0);
glvertex3f(0.5,0.5,-2);
glcolor3f(0,1,1);
glvertex3f(0,0.5,-2);
glend;
```

Das Ergebnis sähe dann so aus:



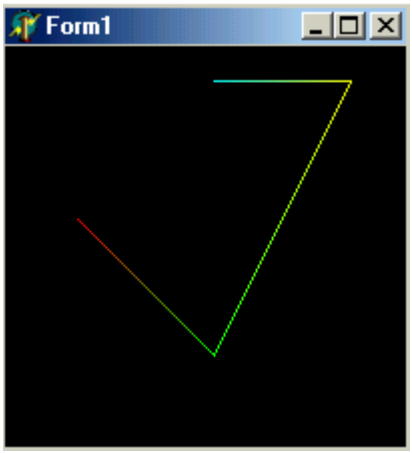
Da hätten wir dann schon mal ein gutes Beispiel für gl_lines: es verknüpft nämlich immer zwei Punkte zu einer Linie... den ersten mit dem zweiten und den dritten mit dem vierten.

Aber es gibt auch noch einen andren Linientypen: „gl_line_strip“. Dieser Parameter nimmt dann immer den 2. Punkt der ersten Linie und zeichnet von dort aus zum dritten, usw. (Die Linien werden aneinandergehängt)

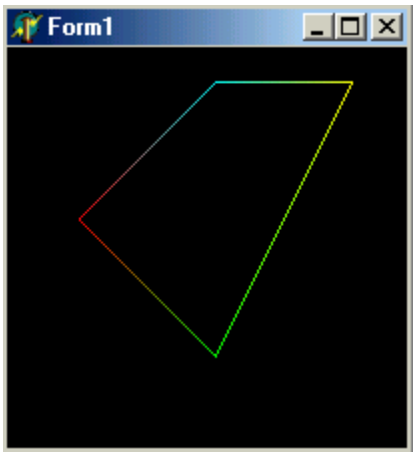
```
glbegin(gl_line_strip);
```

```
glcolor3f(1,0,0);
glvertex3f(-0.5,0,-2);
glcolor3f(0,1,0);
glvertex3f(0,-0.5,-2);
glcolor3f(1,1,0);
glvertex3f(0.5,0.5,-2);
glcolor3f(0,1,1);
glvertex3f(0,0.5,-2);
glend;
```

Und das Bild dazu:



So... und dann gibt es noch eine letzte Linienart: gl_line_loop.
Wie der Source dann aussieht, werde ich nun nicht posten (keine Lust) aber wohl ein Bild, wie unsere Szene dann aussähe:



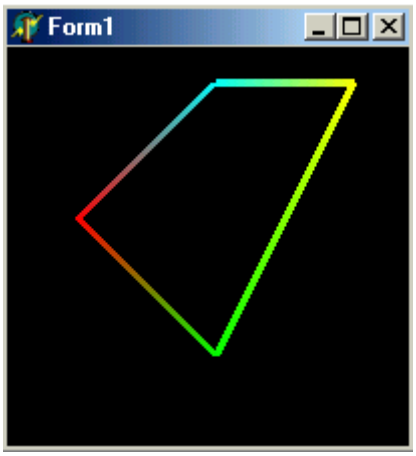
Damit wird klar, was „gl_line_loop“ bewirkt: es ist an sich genau so wie „gl_line_strip“, bis auf, dass es auch den ersten und den letzen Punkt verbindet.

Ach ja: fast hätte ich es vergessen, wo wir doch gerade bei den Linien und Punkten sind:
Es gibt eine Möglichkeit, die Linien und Punktgrößen zu wählen. Dazu muss man vor die ganze Szenerie einfach nur „glLineWidth();“ aufrufen, welches als Parameter die Liniendicke verlangt. Ein Beispiel (ist das von eben):

```
Gllinewidth(4);
glbegin(gl_line_loop);
glcolor3f(1,0,0);
```

```
glvertex3f(-0.5,0,-2);
glcolor3f(0,1,0);
glvertex3f(0,-0.5,-2);
glcolor3f(1,1,0);
glvertex3f(0.5,0.5,-2);
glcolor3f(0,1,1);
glvertex3f(0,0.5,-2);
glend;
```

Das hat dann diese Wirkung:



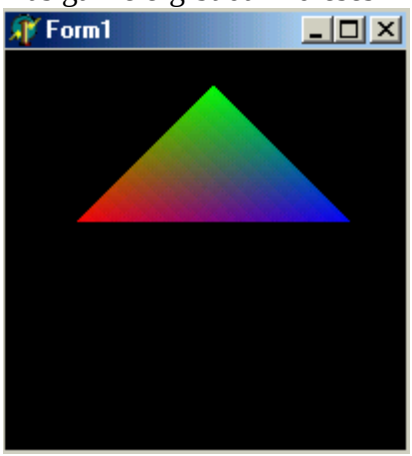
Ach ja: die Punktgrößen kann man wie gesagt auch einstellen: dazu muss man „glPointSize()“ mit dem gewünschten Größenparameter aufrufen.

Von Dreiecken, Dreiecksgruppen und Ventilatoren

So... damit hätten wir Linien und Punkte hinter uns gelassen und widmen uns den ersten Flächenobjekten: Den Dreiecken.
Ein Dreieck wird an sich genauso wie alle anderen Objekte durch Punkte, usw. gezeichnet, der einzige Unterschied ist der Parameter im „Glbegin“. Eine Sache sollte klar sein: ein Dreieck verlangt immer mindestens 3 Punkte (oder ein vielfaches von 3)
Für einfache Dreiecke brauchen wir also den Parameter „gl_triangles“. Dein Beispiel:

```
glbegin(gl_triangles);
glcolor3f(1,0,0);
glvertex3f(-0.5,0,-2); //links
glcolor3f(0,1,0);
glvertex3f(0,0.5,-2);  //oben
glcolor3f(0,0,1);
glvertex3f(0.5,0,-2);  //rechts
glend;
```

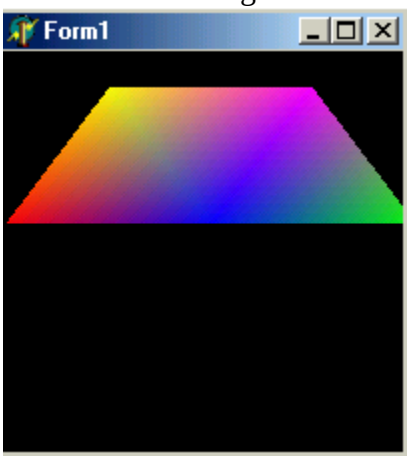
Das ganze ergibt dann dieses Bild:



Außer den normalen Dreiecken gibt es aber auch noch eine 2. Form: „gl_triangle_strip“. Dieses bewirkt, dass immer zwei Dreiecke aneinandergesetzt werden. Dabei wird immer der erste Punkt weggelassen. Das heißt: das erste Dreieck verwendet die Punkte 1,2 und 3. Das zweite die Punkte 2, 3 und 4. das Dritte würde somit die Punkte 3, 4 und 5 verwenden, usw. Auch hier habe ich ein Beispiel gemacht:

```
glbegin(gl_triangle_strip);
glcolor3f(1,0,0);
glvertex3f(-0.75,0,-2);      //1
glcolor3f(1,1,0);
glvertex3f(-0.37,0.5,-2);   //2
glcolor3f(0,0,1);
glvertex3f(0,0,-2);           //3
glcolor3f(1,0,1);
glvertex3f(0.37,0.5,-2);   //4
glcolor3f(0,1,0);
glvertex3f(0.75,0,-2);      //5
glend;
```

Mit dieser Wirkung:

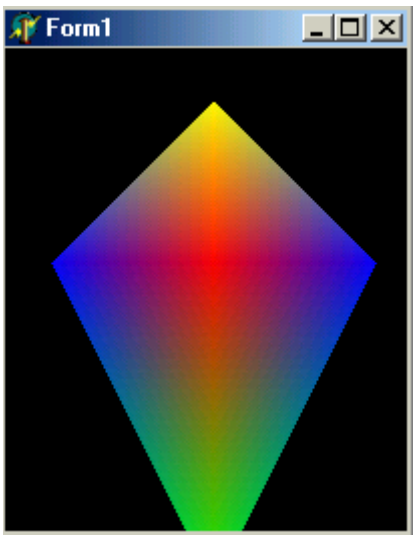


Und was hat das ganze nun mit einem Ventilator zu Tun?? Nun ja: Die Entwickler haben den letzen Dreiecksbefehl „gl_triange_fan“ getauft, der übersetzt soviel wie Gl Dreiecks Ventilator heißt. Nun, was bringt uns dieser Befehl: er ist an sich genauso, wie der „gl_triangle_strip“, bloß, dass immer der 2. Punkt wegfällt. Das bewirkt, dass die ganze Szene

sich quasi um den Ersten Punkt herum aufbaut. Bevor ihr nun aufgebt und sagt, das ihr euch das nicht vorstellen könnt, bringe ich lieber das Sample:

```
glbegin(gl_triangle_fan);
glcolor3f(1,0,0);
glvertex3f(0,0,-2);          //1
glcolor3f(1,1,0);
glvertex3f(0,0.5,-2);         //2
glcolor3f(0,0,1);
glvertex3f(0.5,0,-2);         //3
glcolor3f(0,1,0);
glvertex3f(0,-1,-2);          //4
glcolor3f(0,0,1);
glvertex3f(-0.5,0,-2);        //5
glcolor3f(1,1,0);
glvertex3f(0,0.5,-2);         //6
glend;
```

Womit wir uns mit nur 6 Punkten einen kleinen Drachen gebaut haben (ich weiß, die Punkte 2 und 6 sind identisch, da es leider kein „gl_triagnle_fan_loop“ gibt. An sich sind es daher nur 5 Punkte):



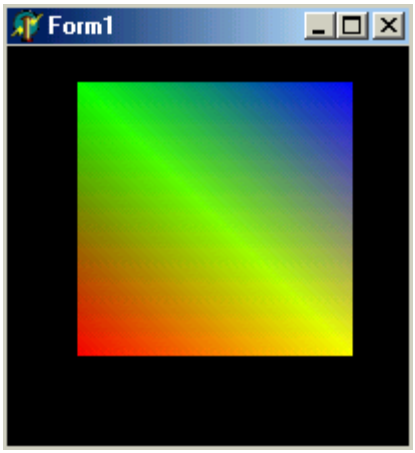
Viereckig geht’s doch auch!

So, nun kommen wir zur Gruppe der Quadrate *puh* ich muss schon sagen: wir haben schon eine Menge geschafft. Die normalen Vierecke sind ja auch nicht allzu schwer... wir haben im letzen Kapitel ja schon eins gesehen. Aber ich will dennoch noch eins bringen:

```
glbegin(gl_quads);
glcolor3f(1,0,0);
glvertex3f(-0.5,-0.5,-2);
glcolor3f(0,1,0);
glvertex3f(-0.5, 0.5,-2);
glcolor3f(0,0,1);
glvertex3f(0.5, 0.5,-2);
glcolor3f(1,1,0);
```

```
glvertex3f(0.5,-0.5,-2);
glend;
```

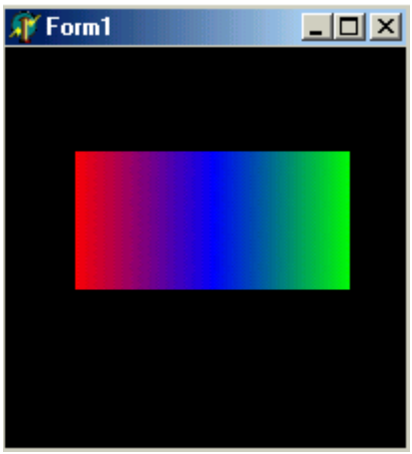
Die Verteilung der Pixel ist an sich auch klar: wir picken uns einen Punkt heraus und beschreiben dann eine Art runde (ob mit oder gegen den Uhrzeigersinn mach zwar schon ne rolle, aber die brauchen wir bislang nicht beachten)
Das sehr erstaunlich Bild *g*:



Aber neben dieser Variante gibt es auch noch eine weitere (Schock schwere Not):
Man kann nämlich auch Vierecke aneinander hängen. Hierzu gibt es den Befehl „gl_quad_strip“. Ein grober Unterschied: die Reihenfolge der Punkte ändert sich! Nun ist es nämlich so, dass die Punkte 1,3,5, usw. und die Punkte 2,4,6, usw. jeweils auf einer Linie zu liegen haben: nichts mehr mit einmal Rumrum! Dabei werden immer die ersten 2 Pixel weggelassen, das heißt, dass das erste Quadrat durch die Punkte 1,2,3 und 4 und das zweite durch die Punkte 3,4,5 und 6 gebildet wird. Ein Sample:

```
glbegin(gl_quad_strip);
glcolor3f(1,0,0);
glvertex3f(-0.5,-0.25,-2);
glvertex3f(-0.5, 0.25,-2);
glcolor3f(0,0,1);
glvertex3f(0,-0.25,-2);
glvertex3f(0, 0.25,-2);
glcolor3f(0,1,0);
glvertex3f(0.5,-0.25,-2);
glvertex3f(0.5, 0.25,-2);
glend;
```

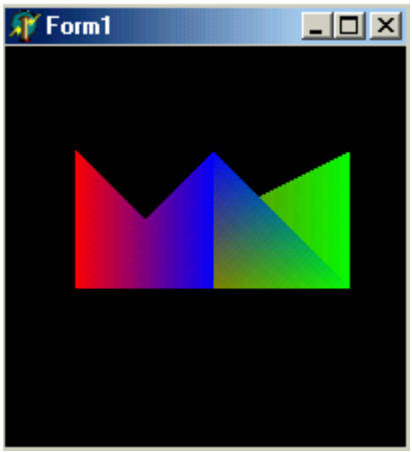
Und ein Bild:



Was man hier nun leider nicht gut sieht:
Die Szene besteht aus 2 Vierecken, die direkt aneinander sitzen. Dabei sind die Punkte 1 (unten links) und 2 (oben links) beide rot, die Punkte 3 (mitte unten) und 4(mitte oben) blau und die beiden letzen (5 unten rechts und 6 oben rechts) grün.

Noch mehr Ecken?

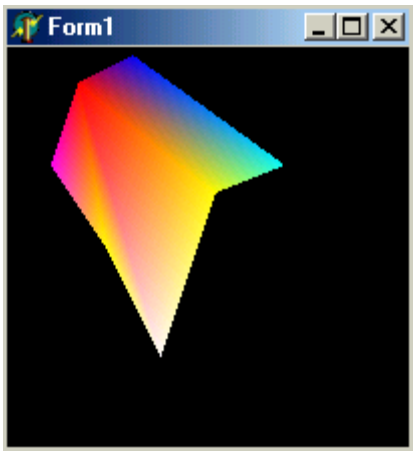
Ja logo. Nun fehlt an sich noch eine Sorte für sich: Die Polygone. Polygone sind schlicht und einfach Vielecke, welche beliebig viele Ecken haben können. Das heißt: alle Punkte zwischen Glbegin und Glend werden mit dem Parameter „gl_polygon“ zu einem einzigen Großem Objekt verflochten. Das heißt natürlich auch, das diese Objekte sehr komplex werden können und damit auch sehr aufwendig zu berechnen werden können. Ein an sich sehr lustiges Beispiel: ändert doch einfach mal das gl_quad_strip – Sample in ein Polygon ab! Dann habt ihr ein nettes kleines Faltschiffchen:



Das ganze ist so zu erklären: Gl_polygon verbindet wieder nach der Reihe, baut aber auch einige Zwischenverbindungen. Dies ist natürlich mehr oder weniger ein Zufallsprodukt gewesen, da ich selber nur mit dem Parameter rumgespielt habe. Hier kommt das richige Sample:

```
glbegin(gl_polygon);
glcolor3f(1,0,0);
glvertex3f(-0.5,0.5,-2);
glcolor3f(0,0,1);
glvertex3f(-0.3,0.6,-2);
glcolor3f(0,1,1);
glvertex3f(0.25,0.2,-2);
glcolor3f(1,1,0);
glvertex3f(0,0.1,-2);
glcolor3f(1,1,1);
glvertex3f(-0.2,-0.5,-2);
glcolor3f(1,1,0);
glvertex3f(-0.4,-0.1,-2);
glcolor3f(1,0,1);
glvertex3f(-0.6,0.2,-2);
glend;
```

Ich muss zwar sagen, dass mir das Schiffchen besser gefiel, als dieses, aber ich wollte mal zeigen, was Ein Polygon alles sein kann (wie merkwürdig es aussehen kann).



Durch den beabsichtigten Knick sieht das ganze sogar schon 3d aus, ist es aber noch nicht! Die Z – Koordinate ist überall gleich! Na ja: echtes 3d kommt nun.

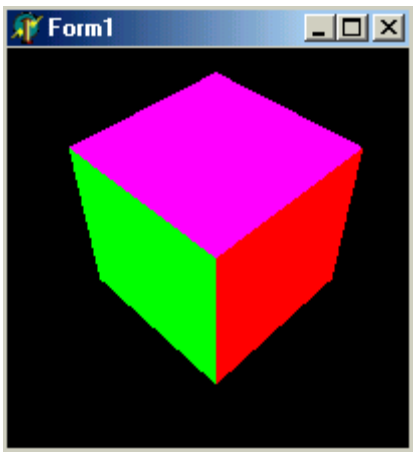
Die Dritte Dimension

Wer dachte, nach 8 Seiten 2d sei nun schon Ende? Nein: ich habe noch einiges mehr im Gepäck (dieser Teil wird wahrscheinlich der längste von allen)
Nun wollen wir uns um 3d – Objekte kümmern. Da will ich zunächst die Pyramide und den Würfel als Beispiele bringen. Also: wie haben wir und einen Würfel vorzustellen? An sich ist ein Würfel nichts andres als 6 Zusammengesetze Quadrate. Und so wird er in Open Gl auch programmiert. Ein Beispiel:

```
gltranslate(0,0,-3);
glrotate(45, 1, 0, 0);
glrotate(45,0,1,0);

glbegin(gl_quads);
glcolor3f(1,0,0);          //Vorderseite (rot)
glvertex3f(-0.5,-0.5,0.5);
glvertex3f(0.5,-0.5,0.5);
glvertex3f(0.5,0.5,0.5);
glvertex3f(-0.5,0.5,0.5);

glcolor3f(0,0,1);          //Rückseite (blau)
glvertex3f(-0.5,-0.5,-0.5);
glvertex3f(0.5,-0.5,-0.5);
glvertex3f(0.5,0.5,-0.5);
glvertex3f(-0.5,0.5,-0.5);

glcolor3f(1,1,0);          //Rechte Seite (gelb)
glvertex3f(0.5,-0.5,0.5);
glvertex3f(0.5,-0.5,-0.5);
glvertex3f(0.5,0.5,-0.5);
glvertex3f(0.5,0.5,0.5);
```

```
glcolor3f(0,1,0);          //Linke Seite (grün)
glvertex3f(-0.5,-0.5,0.5);
glvertex3f(-0.5,-0.5,-0.5);
glvertex3f(-0.5,0.5,-0.5);
glvertex3f(-0.5,0.5,0.5);

glcolor3f(0,1,1);          //Boden (türkis)
glvertex3f(-0.5,-0.5,0.5);
glvertex3f(0.5,-0.5,0.5);
glvertex3f(0.5,-0.5,-0.5);
glvertex3f(-0.5,-0.5,-0.5);

glcolor3f(1,0,1);          //Deckel (lila)
glvertex3f(-0.5,0.5,0.5);
glvertex3f(0.5,0.5,0.5);
glvertex3f(0.5,0.5,-0.5);
glvertex3f(-0.5,0.5,-0.5);
glend;
```

Eines kann man hier sehr deutlich sehen: Ich bevorzuge es, das 3d – Objekt direkt um den Ursprung herum zu zeichnen und dann per gltranslate die ganze Geschichte zu verschieben. Dieses mache ich, da ich nachher das ganze auch noch drehen will.
Drehen? Wieso drehen? Nun ja: wenn ihr die Szene nun startet, so seht ihr nur ein normales rotes Viereck! Besser: ihr seht nur die Vorderseite, was daran liegt, das wir praktisch direkt draufglotzen! Das können wir aber ändern: schreibt nach das gltranslate einfach einmal „glrotate(45, 1, 1, 0);“ Und euerer Würfel (nun auch als Würfel zu erkennen) sollte so aussehen:

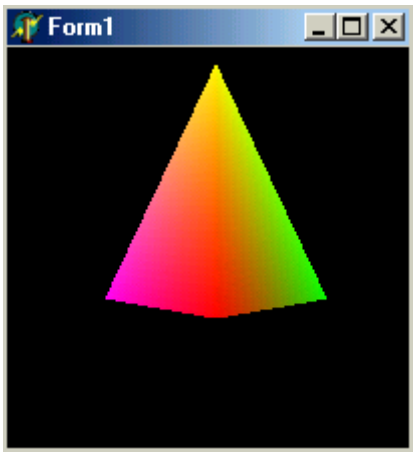


Ist die Lila Oberseite nicht herrlich? Nein! Sieht total schwul aus, ist dem Open Gl – Würfel aber an sich auch egal! Aber wenn der denken könnte...

Nun ja: kommen wir zu unserer Pyramide! Wie haben wir und die denn so vorzustellen? An sich ja bloß als eine Quadratische Grundplatte, auf die drei Dreiecke angekleistert sind. Und wie machen wir das am einfachsten? Wir bauen ein Viereck und verwenden dann „gl_triangle_fan“, da sich ja alle Dreiecke einen Punkt teilen. Damit haben wir einen riesigen Vorteil: wir müssen uns nur die Koordinaten für die Grundfläche ausdenken (können wir von dem Würfel übernehmen) und dann brauchen wir nur noch eine Höhe, wenn wir wieder vom Ursprung ausgehen! Das könnte dann so aussehen:

```
gltranslate(0,0,-3.5);
glrotate(45,0,1,0);

glbegin(gl_quads);
glcolor3f(0,0,1);           //Boden (blau)
glvertex3f(-0.5,-0.5,0.5);
glvertex3f(0.5,-0.5,0.5);
glvertex3f(0.5,-0.5,-0.5);
glvertex3f(-0.5,-0.5,-0.5);
glend;

glbegin(gl_triangle_fan);
glcolor3f(1,1,0);     //Die Spitze (gelb)
glvertex3f(0,1,0);

glcolor3f(1,0,0);     //Punkt 2 = Punkt 6 (rot)
glvertex3f(-0.5,-0.5,0.5);

glcolor3f(0,1,0);     //Punkt 3 (grün)
glvertex3f(0.5,-0.5,0.5);

glcolor3f(0,1,1);     //Punkt 4 (türkis)
glvertex3f(0.5,-0.5,-0.5);

glcolor3f(1,0,1);     //Punkt 5 (lila)
glvertex3f(-0.5,-0.5,-0.5);

glcolor3f(1,0,0);     //Punkt 6 = Punkt 2 (rot)
glvertex3f(-0.5,-0.5,0.5);

glend;
```

Tja... und oh wunder das Aussehen:



Nachwort

So... das war’s auch „schon“. Eigentlich wollte ich noch etwas über runde Objekt d.h. über Kugeln bringen, aber ich habe festgestellt, dass dieses wenig Sinn macht, da alle runden 3d Objekte wie Scheiben wirken, solang wir noch keine Lichter in unserer Szene haben. Ich verspreche aber, dass ich alles nachholen werde, sobald ich das Lichter - Kapitel mit euch gemacht habe. Damit wären wir mit den „Zeichenstiften“ von Open Gl auch schon fertig und ich hoffe euch in zwei Wochen wiederbegrüßen zu dürfen, wenn wir uns das Thema „Texturen“ vornehmen (ß Boa... großes Ziel *g*) Na ja: Bis dann den! Und: schmeißt schon mal euer Paintshop oder Photoshop an! Wir werden es brauchen...

Mr_T