Open Gl - Delphi – Kurs Teil 13: Partikeleffekte

Einleitung:

So, nach langem hin und her, welches Thema denn nun in diesem 13. Teil kommen sollte (es gab bereits die ersten Seiten zu einem anderen Thema, welches dann aber in den nächsten Praxisteil verlegt wurde) und am Ende kam denn Aufgrund aktueller Ereignisse dieses Thema, welches sich mir irgendwie aufdrängte, auch wenn es erst viel später im Kurs geplant war: Partikel.
Was wäre eine modernes Computerspiel ohne Partikel? Eben: Ziemlich effektarm und unnatürlich würde es wirken! Und deshalb befassen wir uns in diesem Teil damit eine einfache Partikel-Engine zu basteln, mit welcher wir recht schnell ganz nette Effekte herstellen können.

Was sind Partikel?

Zu allererst sollten wir uns natürlich fragen, was Partikel genau sind. Dieses ist leicht zu beantworten: Partikel sind sehr kleine Objekte, welche Physisch sich manipulieren lassen und damit sehr realitätsgetreu wirkende Effekte wie Wasser, Feuer oder auch Rauch bilden können. Meistens sehen Partikeleffekte total chaotisch aus, d.h. wie in der Wirklichkeit vermag man oftmals nicht wirklich einen Gesamtzusammenhang für das Verhalten aller Partikel zu finden, auch wenn jedes einzelne Partikel sich physikalisch durchaus korrekt verhält.
Dies ist auch der Ansatz hinter einer Partikelengine:
An sich werden nur sehr viele Partikel (d.h. kleine Teile) erstellt, welche dann durch fest vorgeschriebene Routinen von Frame zu Frame ihrer Physik nach modifiziert werden.

Also los!

Also was brauchen wir nun um unsere eigne kleine Partikelengine zu basteln? Nun, normalerweise würde man nun Anfangen, sich einfach ein Array mit Koordinaten von Punkten anzulegen und dies als Darstellungsgrundlage zu verwenden. An sich ist die Idee auch nicht blöd, hat sie aber dennoch einen Haken: damit schafft man sich nur die Grundlage für einen einzigen Effekt... nicht wirklich Praktikabel also, wenn man wirklich so was wie eine kleine „Engine“ basteln möchte.
Also beginnen wir mit einem Record, welcher den Basistyp für jedes Partikel darstellen wird:

```
type tpartikel = record                    //Der Partikel - Typ
    position : array [0..2] of double;     //Die Position des Partikels
    start_position : array [0..2] of double;//Die Startposition
    rotation : integer;                    //Die "Richtung" in der horizontalen Bewegung
    hor_speed : double;                    //Die dazugehörige Geschwindigkeit
    vert_speed : double;                   //Die auf- oder abwärts gerichtete Geschwindigkeit
    color : array [0..2] of double;        //Die Farbe des Partikels
    age : double;                          //Das Alter des Partikels
    active : boolean;                      //Ist der Partikel Aktiv?
    end;
```

Soweit doch noch ganz beschaulich, oder? Zumindest sollte es nicht schwer fallen, nachzuvollziehen, wofür die Eigenschafen alle so gut sind (wozu schreib ich Kommentare?).

So, nu haben wir also eine art Container für unsere Partikel (besser: wir haben einen Partikel-Basistyp), aber was machen wir nun damit? Nun, ohne einen Gesamteffekt macht so ein Partikel nun echt nicht viel her ... also muss noch ein weiteres Record für die Effekte her:

```
type teffekt = record                              //Der "Rahmen" für unsere Effekte
   start_position : array [0..2] of double;        //Die Position des Mittelpunktes

   vert_speed_range : array [0..1] of double;  //Der Bereich der "Speed" für die Vertikale
   hor_speed_range : array [0..1] of double;   //Das selbe nochmal für die Horizontale
   rotation_range : array [0..1] of integer;
   //Der Bereich für die Rotation (horizontal) in der Flugrichtung;

   partikel_count : integer;              //Die Anzahl an Partikeln in diesem Effekt
   partikel : array of Tpartikel;         //Das eigentliche Partikel-Array

   start_color : array [0..2] of double; //Die Startfarbe aller Partikel (Alter = 0)
   end_color : array [0..2] of double;  //Die Endfarbe aller Partikel (Alter = max_age)
   max_age : double;                     //Das maximale Alter eines Partikels (in Sekunden)
   gravity : double;                     //Wie stark wirkt die Schwerkraft auf diese Partikel?
   part_size : double;                   //Größe der Partikel des Effekts
   end;
```

Soweit war das ganze doch noch nicht schlimm, oder?
An sich sollte auch jeder dazu in der Lage sein nachzuvollziehen, wozu die einzelnen Eigenschaften der Partikel bzw. des Effektes so gut sind. Sollte irgendwer Verständnisprobleme bezüglich der „horr_speed_range“ und der „rotation“ haben, dem sollte diese Anmerkung vielleicht behilflich sein:
Unsere kleine Partikelengine ist in diesem Falle darauf ausgelegt, hauptsächlich Effekte zu erzeugen, bei denen die Partikel mehr oder weniger eine kreisförmige Flugrichtung von dem Ausgangspunkt weg haben. Dieses schränkt die Verwendung des ganzen zwar ziemlich ein, aber dafür ist die Partikel-Physik leichter zu berechnen. Der Parameter „rotation_range“ gibt nun an, in welchem „Winkelbereich“ von der Startposition aus die Partikel wegfliegen können, damit die Partikel nicht immer in einem Vollkreis davonfliegen.

So, nun haben wir also unsere Records, in denen wir Effekte bzw. die dazugehörigen Partikel kapseln können. Was nun noch fehlt sind Prozeduren, welche die Partikel auf den Schirm bringen, neue erschaffen und die vorhandene bewegen, ansonsten hätte das alles hier wohl reichlich wenig Sinn (könnte ich mir so denken *ggg*).

Also ans Werk. Zunächst wollen wir eine Prozedur basteln, welche neue Partikel in das Effekt-Array einfügt (ich weiß, das ist etwas mehr Source...):

```
procedure add_new_partikel(var effekt : teffekt);
var
i, v : integer;
help_ext : double;
help_int : integer;
```

```
begin
i := floor((effekt.partikel_count * timefactor)/ effekt.max_age);
//Anzahl der neuen Partikel berechnen
if i = 0 then
i := 1;
v := 0;
repeat
 if effekt.partikel[v].active = false then //Ist der Partikel frei?
 begin                                     //Ja --> Besetzen
 dec(i);
  with effekt.partikel[v] do
  begin
  active := true;                                  //Dieser Partikel wird nun verwendet
  start_position[0] := effekt.start_position[0]; //Die Startposition setzen
  start_position[1] := effekt.start_position[1];
  start_position[2] := effekt.start_position[2];

  //Die Speed in der Vertikalen berechnen
  help_ext := effekt.vert_speed_range[1] - effekt.vert_speed_range[0];
   if help_ext > 0 then
   begin
   help_ext := help_ext*1000;
   help_int := random(floor(help_ext));
   vert_speed := effekt.vert_speed_range[0] + (help_int/1000);
   end
   else
   vert_speed := effekt.vert_speed_range[0];

   //Die Speed in der Horizontalen berechnen
   help_ext := effekt.hor_speed_range[1] - effekt.hor_speed_range[0];
    if help_ext > 0 then
    begin
    help_ext := help_ext*1000;
    help_int := random(floor(help_ext));
    hor_speed := effekt.hor_speed_range[0] + (help_int/1000);
    end
    else
    hor_speed := effekt.hor_speed_range[0];

   //Wir berechnen den Winkel der Flugrichtung
   help_int := effekt.rotation_range[1] - effekt.rotation_range[0];
   help_int := random(help_int);
   rotation := effekt.rotation_range[0] + help_int;

   color[0] := effekt.start_color[0];      //Nun noch die Startfarbe setzen
   color[1] := effekt.start_color[1];
   color[2] := effekt.start_color[2];

   age := 0;                                //Alter = 0, weil ja neuer Partikel
 end;
end;
```

```
inc(v);
until (i = 0) or (v=effekt.partikel_count);                    //Bis wir genug Partikel haben
end;
```

Ufi ... auf den ersten Blick sieht das nun sicherlich wie der blanke Horror in Code-Forum aus, aber wozu ist das hier ein Tutorial? *ggg*

Wollen wir mal sehen, was der Code so hergibt:

Also das erste, was wir machen ist die Anzahl an Partikel ausrechnen, welche wir in diesem Render-Durchgang erstellen müssen. Diese Zahl wird zunächst mal in „i“ gespeichert.
(i := floor((effekt.partikel_count * timefactor)/ effekt.max_age);)

Denn Starten wir eine kleine Repeat-Schleife, in welcher wir unsere Variable „V“ immer um eines erhöhen. „V” verwenden wir nun als Zählvariable, mit deren Hilfe wir jedes einzelne Partikel des Effektes einzeln durchgehen und prüfen, ob dieses noch nicht benutzt ist, also ob wir es verwenden können. Sollte dieses der Fall sein, so wird „i“ um eins erniedrigt (ein Partikel weniger, welches wir adden müssen). Die Schleife läuft nun so lange ab, bis wir entweder genug neue Partikel haben (i = 0) oder wir zu viele Gesamtpartikel bekommen könnten (v=effekt.partikel_count).

Innerhalb unserer kleinen Schleife weisen wir nun zunächst unserem neuen Partikel seine Startposition zu ... Ordnung muss sein.
Denn wird dem Partikel eine Geschwindigkeit in der Vertikalen zugeordnet. Diese wird dadurch berechnet, dass wir zunächst in eine Hilfs-Fließkommazahl (help_ext) die Differenz der oberen und unteren Grenze für die Geschwindigkeit einlesen. Ist dieser Wert positiv, so wird er mit 1000 multipliziert und denn in einen Integer um gewandelt, wobei wir gleichzeitig die Random-Funktion auf diesen neuen Integer anwenden. Unserer kleiner Hilfsinteger (help_int) hat nun einen beliebigen Wert zwischen 0 und dem 1000 fachen der Differenz zwischen oberer und unterer Grenze. Wenn wir nun also unseren Integer wieder durch 100 teilen und die untere Grenze hinzuaddieren bekommen wir einen „Random“ – Wert zwischen unterer und oberer Grenze, und genau das machen wir auch. Das Ergebnis wird der Eigenschaft „vert_speed“ unseres Partikels zugeordnet.

An sich genau dasselbe wird nun auch noch mit der horizontalen Fluggeschwindigkeit und dem Flugwinkel gemacht, wobei beim Flugwinkel wir einfach mit ganzen Zahlen rechnen können und damit unsere Fließkommazahl nicht brauchen.

Zu allerletzt wird denn noch die Farbe auf den Startwert gesetzt und das Alter des partikels auf 0 gesetzt. Fertig ist unserer neuer Partikel ;-)

So, damit können wir unser Array schon mal ganz gut mit Partikel füllen (bei wem der Source nicht geht: um den Effekt auf allen Rechnern gleich schnell ablaufen zu lassen habe ich die Variable „timefactor“ eingebaut .... die muss denn in eurer Version auch her)

So, denn wollen wir die Partikel aber man auch zwischen den Frames bewegen:

```
procedure handle_partikel(var effekt : teffekt);
var i : integer;
begin
 for i := 0 to effekt.partikel_count-1 do
```

```
 begin

  if effekt.partikel[i].age >= effekt.max_age then   //Ist der Partikel zu alt?
  effekt.partikel[i].active := false;                //Ja --> wir werfen ihn raus :-)

  if effekt.partikel[i].active = true then
  begin
   effekt.partikel[i].position[0] := effekt.partikel[i].start_position[0] +
(sin(degtorad(effekt.partikel[i].rotation)) * effekt.partikel[i].hor_speed *
effekt.partikel[i].age);

   effekt.partikel[i].position[1] := effekt.partikel[i].start_position[1] - (effekt.gravity * 0.5  *
effekt.partikel[i].age * effekt.partikel[i].age) + (effekt.partikel[i].vert_speed *
effekt.partikel[i].age);

   effekt.partikel[i].position[2] := effekt.partikel[i].start_position[2] +
(cos(degtorad(effekt.partikel[i].rotation)) * effekt.partikel[i].hor_speed *
effekt.partikel[i].age);

   effekt.partikel[i].color[0] := (effekt.start_color[0] * (1 - (effekt.partikel[i].age /
effekt.max_age))) +  (effekt.end_color[0] * (effekt.partikel[i].age / effekt.max_age ));

   effekt.partikel[i].color[1] := (effekt.start_color[1] * (1 - (effekt.partikel[i].age /
effekt.max_age))) +  (effekt.end_color[1] * (effekt.partikel[i].age / effekt.max_age ));

   effekt.partikel[i].color[2] := (effekt.start_color[2] * (1 - (effekt.partikel[i].age /
effekt.max_age))) +  (effekt.end_color[2] * (effekt.partikel[i].age / effekt.max_age ));

   effekt.partikel[i].age := effekt.partikel[i].age + timefactor;
  end;

 end;
end;
```

Ach du *******. Was ist das denn nu?
Naja, es ist gar nicht so schlimm, wie es aussieht.
Zunächst werden alle Alten Partikel erstmal Aussortiert (alles hat mal n Ende...) und dann werden die Verbleibenden anhand ihrer Geschwindigkeits-Daten an ihre neue Position verfrachtet. Zu guter letzt wird noch die Farbe des Partikels ganz einfach anhand seines Alters zwischen Anfangs und Endwert interpoliert.
Sollte irgendjemand noch Probleme mit dem Verändern der Position haben, denn bitteich ihn drum ein Physik-Buch der 11. Klasse aufzusuchen (bei uns ist es zumindest Stoff der 11.) ... habe keine Lust das zu erklären ;-)

Nun müssen unsere kleinen Partikel also noch gezeichnet werden.
Das ist nun aber auch keine Kunst mehr. Farbe, Position, usw. haben wir alles zusammen, also auf geht’s:

```
procedure draw_partikel(effekt : teffekt);
var
i : integer;
begin
glbegin(gl_quads);
for i := 0 to effekt.partikel_count-1 do
begin
 if effekt.partikel[i].active = true then
 begin
  glcolor3f(effekt.partikel[i].color[0], effekt.partikel[i].color[1], effekt.partikel[i].color[2]);

  glvertex3f(effekt.partikel[i].position[0]-effekt.part_size,
effekt.partikel[i].position[1]+effekt.part_size, effekt.partikel[i].position[2]);

  glvertex3f(effekt.partikel[i].position[0]-effekt.part_size, effekt.partikel[i].position[1]-
effekt.part_size, effekt.partikel[i].position[2]);

  glvertex3f(effekt.partikel[i].position[0]+effekt.part_size, effekt.partikel[i].position[1]-
effekt.part_size, effekt.partikel[i].position[2]);

  glvertex3f(effekt.partikel[i].position[0]+effekt.part_size,
effekt.partikel[i].position[1]+effekt.part_size, effekt.partikel[i].position[2]);

 end;
end;
glend;
end;
```

So, das war nun wirklich nicht mehr schwierig. An sich machen wir nichts anderes, als nur In einer Schleife alle Partikel des Effektes zu durchlaufen und sollte dieser Partikel „active“ sein, diesen mit der entsprechenden Farbe auf den Schirm zu bringen.

Das Billboarding

So, kommen wir zur letzen kleinen Technik hier, welche wir noch für unsere Effektmaschienerie brauchen: Das Billboarding. Unter Billboarding versteht man an sich, das Objekte immer so gedreht werden, dass sie genau platt zu dem Betrachter zeigen, also dass man die Kanten der Objekte nicht sehen kann. Diese Technik müssen wir auch verwenden, weil wir ja (an sich um Rechenpower zu sparen) nur Quads, aber keine Würfel als Partikel zeichnen, unsere Szene aber sicherlich nicht nur von Vorne Betrachten wollen.
Das Billboarding an sich läuft nun so ab, als das man alle Werte, welche nichts mit der Verschiebung zu tun haben, aus der Darstellungsmatrix rauswirft, also jegliche Art von Rotation verloren geht.
An dieser Stelle möchte ich noch erwähnen, dass die zwei nun folgenden Quelltexte nicht von mir sind, sondern das ich diese im DGL-Forum gefunden habe.

Das Billboarding „aktiviert“ man praktisch so:

```
procedure BillboardBegin;
var x,y : byte;
    Matrix : array[0..15] of single;

begin
//save original Matrix
 glPushMatrix;
 glGetFloatv(GL_MODELVIEW_MATRIX, @Matrix);
 for x := 0 to 2 do
  for y := 0 to 2 do
   if x=y then Matrix[x*4+y] := 1 else Matrix[x*4+y] := 0;
 glLoadMatrixf(@Matrix);
end;
```

Und so wird man es (sehr einfach) wieder los:

```
procedure BillboardEnd;
begin
  //restore original Matrix
  glPopMatrix;
end;
```

So, damit haben wir alle Vorbereitungen abgeschlossen ;-)

Lasset es regnen :-)

So, Zeit also unsere Effektmaschinerie einmal zu testen.
Das erste, was wir nun machen, ist, unsere Render-Prozedur erstmal mit Inhalt zu füllen:

```
BillboardBegin;
handle_partikel(springbrunnen);
draw_partikel(springbrunnen);
add_new_partikel(springbrunnen);
BillboardEnd;
```

Wie ihr seht, haben wir einen Effekt mit Namen „Springbrunnen“. Um diesen überhaupt verwenden zu können, sollten wir zunächst mal natürlich diese (globale) Variable anlegen:

```
...
springbrunnen : teffekt;
...
```

So, wenn wir nun aber unsere Anwendung in Freudiger Erwartung starten passiert rein gar nichts. Noch viel schlimmer: es dürfte ne Fehlermeldung geben. Den Grund dafür kann und sollte man darin suchen, dass wir noch keinerlei Parameter an unseren Effekt weitergegeben haben, dieser also gar nicht weiß, wie er aussehen soll.
Also tun wir dieses, indem wir ein wenig Source unten an unsere OnCreate – Prozedur anhängen:

```
with springbrunnen do
```

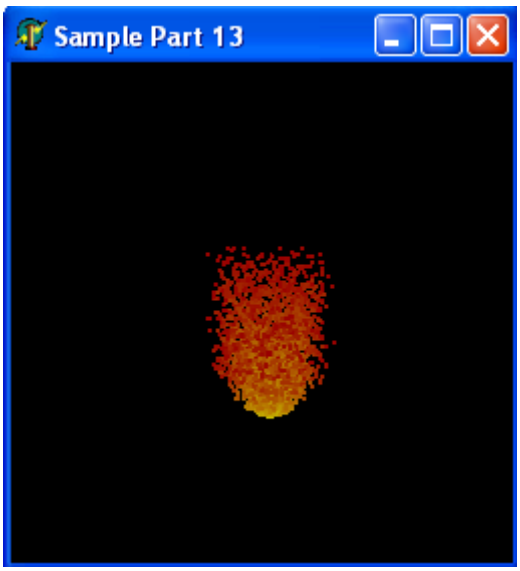
```
begin
 start_position[0] := 0;
 start_position[1] := -1;
 start_position[2] := -5;

 vert_speed_range[0] := 4;
 vert_speed_range[1] := 4;
 hor_speed_range[0] := 0.8;
 hor_speed_range[1] := 1;
 rotation_range[0] := 0;
 rotation_range[1] := 360;

 partikel_count := 5000;
 setlength(partikel,partikel_count);

 start_color[0] := 1;
 start_color[1] := 1;
 start_color[2] := 1;
 end_color[0] := 0.3;
 end_color[1] := 0.3;
 end_color[2] := 0.6;

 max_age := 0.81;
 gravity := 9.81;
 part_size := 0.005;
end;
```

Ufi, das sind aber viele Parameter, welche man einstellen kann. Aber umso besser... Das Ergebnis kann sich auf jeden Falle sehen lassen:

Man kann aber natürlich nicht nur Springbrunnen erzeugen. Auch so etwas, was so ähnlich aussieht wie Feuer (besser habe ich es nicht hinbekommen) ist möglich:

```
with feuer do
begin
 start_position[0] := 0;
 start_position[1] := -1;
 start_position[2] := -5;

 vert_speed_range[0] := 0;
 vert_speed_range[1] := 1.5;
 hor_speed_range[0] := 0.3;
 hor_speed_range[1] := 0.8;
 rotation_range[0] := 0;
 rotation_range[1] := 360;

 partikel_count := 5000;
 setlength(partikel,partikel_count);

 start_color[0] := 0.8;
 start_color[1] := 0.8;
 start_color[2] := 0;
 end_color[0] := 0.6;
 end_color[1] := 0;
 end_color[2] := 0;

 max_age := 0.6;
 gravity := -2;
 part_size := 0.015;
end;
```

Und auch hier ein Bild vom Ergebnis:



Nachwort

So, an dieser Stelle möchte ich mich noch mal herzlichste dafür entschuldigen, dass es mit diesem Kursteil so lange gebraucht hat. Es war wirklich nicht böse gemeint und ich gelobe Besserung ;-)

Zu den Partikeln sei zuletzt noch gesagt, dass es natürlich viele Möglichkeiten gibt, dieses kleine Partikelsystem zu erweitern. Ein nächster Schritt wäre sicherlich das Einbauen des 4. Farbkanals, damit wir auch etwas Blenden können (so komplett solid sehen die Partikel auf Dauer nicht gut aus) und ein weiterer wichtiger Schritt wäre beispielsweise das einbauen von Texturen zur Erhöhung der Effektqualität.
Ich wünsche euch allen noch viel Spaß mit eurem Partikelsystem und hoffe, das ihr nach diesem kleinem einstieg in die Welt der Effekte auch im nächsten Kursteil wieder dabei sein würdet.
Eurer

Wilke Trei (Mr_T)