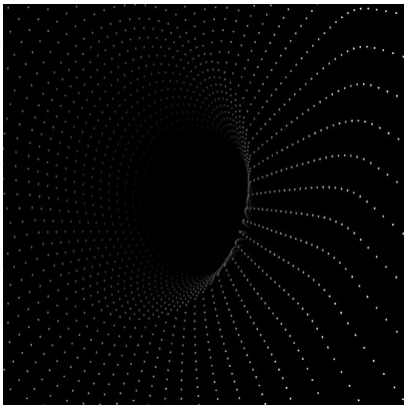


Pixeltunnel im GameMaker

Description

Pixeltunnel sind schon sehr alt. Das bekannteste Beispiel hierfür stammt aus der Demo Second Reality aus dem Jahr 1993. Natürlich lässt sich so ein Effekt auch in GameMaker, mit relativ wenig Aufwand, realisieren.

Das Tutorial ist vor allem für Anfänger gedacht und ist, wie auch schon der Shadebob-Effekt, nicht die performanteste Lösung. Für die meisten Sachen läuft aber auch dieser Effekt sehr schnell. Verwenden können wir den Effekt als Hintergrund bei Spielen, im Intro, bei einem Übergang oder bei Credits.

Der gezeigte Effekt soll zum experimentieren einladen. Du kannst Farbe, Größe und Geschwindigkeit variieren und natürlich auch unterschiedliche Pfade testen.

Der Raum

Zunächst brauchen wir einen Raum. Ich stellte 1024×768 Pixel als Raumgröße ein. Hintergrund Schwarz und Speed 60. So sieht das Ganze dann auch richtig flüssig aus. Den Raum kannst Du noch auf lassen, wir brauchen ihn gleich noch.

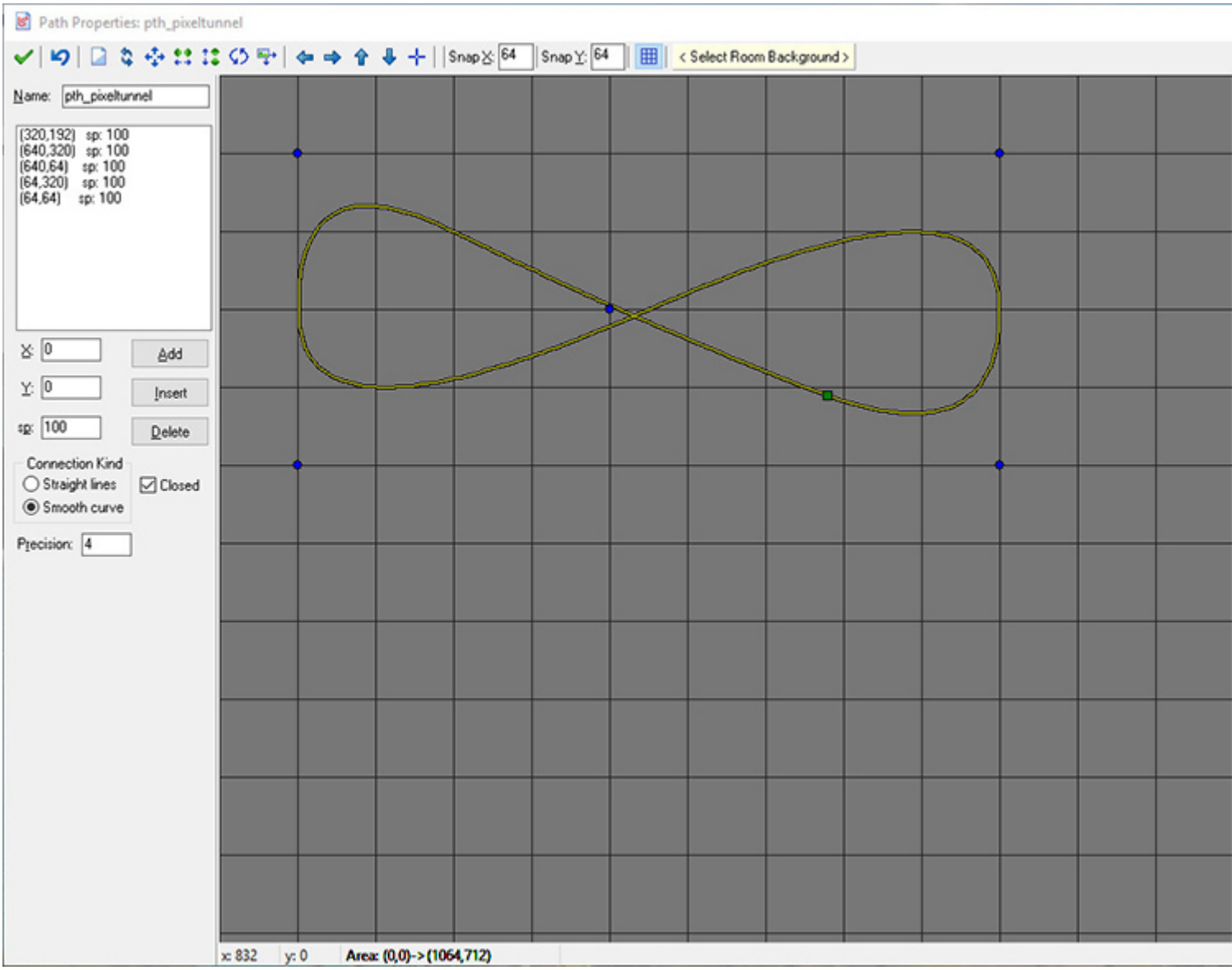
Objekte erstellen

Wir brauchen zwei Objekte: *obj_pixeltunnel* und *obj_pixel*. Wenn Du Dir das Shadebob-Tutorial angeschaut hast, wirst Du es schon ahnen, worauf dieses Tutorial hinaus läuft. Wir haben wieder ein Master-Objekt, nämlich *obj_pixeltunnel* und dieses generiert die Pixel, welche selbständig fliegen.

Wenn Du die beiden Objekte erstellt hast, kannst Du das Objekt *obj_pixeltunnel* in den Raum setzen. Es ist egal wo, bei mir sind solche Objekte immer links oben in der Ecke. Anschließend kannst Du den Raum schließen, wir brauchen ihn nicht mehr.

Pfad erstellen

Damit der Tunnel cool aussieht, lassen wir ihn auf einem Pfad bewegen. Erstelle einen Pfad mit dem Namen *pth_pixeltunnel*. Kreuze **Closed** an und klick auf **Smooth curve**. Somit können wir eine schöne Schleife als Pfad erstellen. Du kannst später natürlich den Pfad ändern und erweitern. Mein Pfad sieht so aus:



Da wir die Pfadkoordinaten relativ benutzen, ist der Startpunkt eigentlich egal. Wenn Du fertig bist, kannst Du den Pfad schließen.

Objekt obj_pixel

Jetzt nehmen wir uns den Pixel vor. Eigentlich wird das kein einzelner Pixel, sondern ein sehr kleiner Kreis. Das hat den Vorteil, dass wir über den Radius jederzeit die Größe verändern können.

Create-Event

```
speed = 4;          // Geschwindigkeit
speedup = 0.04;     // Beschleunigung

// RGB-Werte
colr = 10;
colg = 10;
colb = 10;

// Farbe
col = c_gray;
```

Wir setzen eine Startgeschwindigkeit für den Pixel. Im Beispiel liegt die bei 4. Dann haben wir eine kleine Beschleunigung drin. Das heißt, der Pixel wird, wenn auch nur minimal, immer schneller.

Außerdem wollen wir die Farbe leicht verändern. Am Anfang sind die Pixel etwas dunkler, werden dann aber immer heller. So bekommen wir eine etwas bessere Tiefenwirkung. Da wir mit RGB-Farben arbeiten, brauchen wir hier drei Variablen. Der Startwert liegt bei 10. Im Step-Event werden diese Werte erhöht und die Farbe definiert. Die Variable *col* ist nur da, damit wir sie definiert haben.

Step-Event

```
speed += speedup;   // Der Pixel wird immer schneller

// Wenn sich der Pixel außerhalb des Raumes befindet, zerstört er sich
if (x < 0) || (x > room_width) || (y < 0) || (y > room_height)
{
    instance_destroy();
}

// Der Pixel wird immer heller
if (colr < 254)
{
    colr += 2;
    colg += 2;
    colb += 2;
}

// Definiert die Farbe anhand der neuen RGB-Wert Daten
col = make_colour_rgb(colr, colg, colb);
```

Bekanntlich wird der Step-Event einmal pro Frame ausgeführt. Am Anfang beschleunigen wir den Pixel um den Wert *speedup*, der im Create-Event mit 0.04 definiert wurde. Da wir im Raum als Speed 60 eingestellt haben, erfolgt die Beschleunigung 60 Mal in der Sekunde. Nach einer Sekunde haben wir somit eine Geschwindigkeit von 6.4, nach zwei Sekunden 8.8 und so weiter.

Im zweiten Abschnitt prüfen wir, ob sich der Pixel außerhalb des Raumes befindet. Wenn ja, wird er zerstört. Die Prüfung kann man natürlich auch anpassen und den Tunnel auf einen kleinen Bereich des Bildschirms beschränken, oder es auf den View anpassen.

Zuletzt geht es um die Farbe. Wir begannen im Create-Event mit dem Wert 10. Hier wird der Wert nun mit jedem Schritt um 2 erhöht. Nach ungefähr zwei Sekunden ist der Pixel also ganz weiß. Anschließend werden die neuen Farbwerte mit **make_colour_rgb** an die Variable *col* übergeben. Die brauchen wir dann im **Draw-Event**.

Draw-Event

```
// Der Pixel wird gezeichnet, der Radius kann frei definiert werden.
// Standard = 2

draw_set_alpha(1);
draw_set_color(col);
draw_circle(x, y, 2, 0);
```

Hier wird der Pixel, beziehungsweise der kleine Kreis gezeichnet. Wer möchte, hat hier noch die Möglichkeit mit der Größe und der Transparenz zu spielen.

Objekt obj_pixeltunnel

Wir „binden” das Objekt erst einmal an einen Pfad und dann erzeugen wir im Alarm die Pixel. Für solch einfache Aufgaben sind Alarme als Schleife super geeignet, da wir nur jeden dritten Step die Pixel erzeugen und wir bei Alarmen, im Gegensatz zum Step-Event, nicht mit Hilfsvariablen arbeiten müssen.

Create-Event

```
var i, o;

// Wir verschieben das Objekt in die Mitte des Raumes.
x = room_width / 2;
y = room_height / 2;

// Das Objekt bewegt sich den Pfad entlang
path_start(pth_pixeltunnel, 4, 1, 0);


dd = 0;         // Direction (Richtung)
rad = 120;      // Radius
rad_up = true;  // Radius darf sich erweitern

alarm[0] = 1;               // Im Alarm 0 zeichnen wir die Pixel
```

Die Variablen *i* und *o* brauchen wir im Alarm[0]. Dann platzieren wir das Objekt in der Mitte des Raumes. Mit **path_start** verbinden wir das Objekt mit dem zuvor erstellten Pfad. Dann definieren wir

drei Variablen. Wir haben eine Ausgangsrichtung, einen Radius und die Möglichkeit, den Radius zu verkleinern und zu vergrößern. So wirkt der Tunnel lebendiger. Die Variablen brauchen wir ebenfalls im Alarm.

Alarm[0]-Event

```
// Wir halten die aktuellen Koordinaten fest
xx = x;
yy = y;

// In einer Schleife generieren wir 72 Pixel im Kreis
for (i = 0; i < 72; i += 1)
{
    o = instance_create(xx + lengthdir_x(rad, dd), yy + lengthdir_y(rad, dd),
    o.direction = dd;
    xx += 0;
    dd += 5;    // Abstand = 5 Grad. Anzahl der Pixel multipliziert mit dem Ab
}

// Der Radius erweitert sich oder wird kleiner
if (rad_up)
{
    rad += 4;

    if (rad >= 190)
    {
        rad_up = false;
    }
} else {
    rad -= 4;

    if (rad <= 80)
    {
        rad_up = true;
    }
}

alarm[0] = 0.05 * room_speed;   // Der Alarm wird neu gestartet
```

Das sieht komplizierter aus, als es ist. *xx* und *yy* bekommen die aktuelle Koordinate unseres Objekts, *obj_pixeltunnel*. Darunter haben wir eine **for-Schleife**. Bei jedem Aufruf geht die Schleife 72 Mal durch und erzeugt ein Objekt. Schauen wir uns das genauer an:

```
o = instance_create(xx + lengthdir_x(rad, dd), yy + lengthdir_y(rad, dd), obj_
```

Die Variable *0* brauchen wir, um dem erstellten Objekt noch ein paar Dinge mit auf den Weg zu geben. Mit **instance_create** erstellen wir das Objekt. Dafür brauchen wir die Koordinaten *x*, *y* und das Objekt selbst, also *obj_pixel*. Wie Du sehen kannst, haben wir nicht nur X und Y angegeben, sondern auch den Befehl **lengthdir_x** bzw. **lengthdir_y**. Um zu verstehen, warum wir diese beiden Befehle nutzen, muss man erst einmal das Problem verstehen. Im Prinzip zeichnen wir einen Kreis oder genauer gesagt, den Umfang eines Kreises. Der Befehl **instance_create** will aber von uns zwei Koordinaten haben, die wir errechnen müssen. Diese Werte, für X und Y, geben uns die Befehle **lengthdir_x** und **lengthdir_y**

. Wir müssen lediglich den Radius und die Richtung angeben und erfahren, wo die Koordinate liegt. Diesen Wert addieren wir zu *xx* bzw. *yy* und haben genau den Punkt, den wir brauchen. GameMaker greift und hier wieder mächtig unter die Arme.

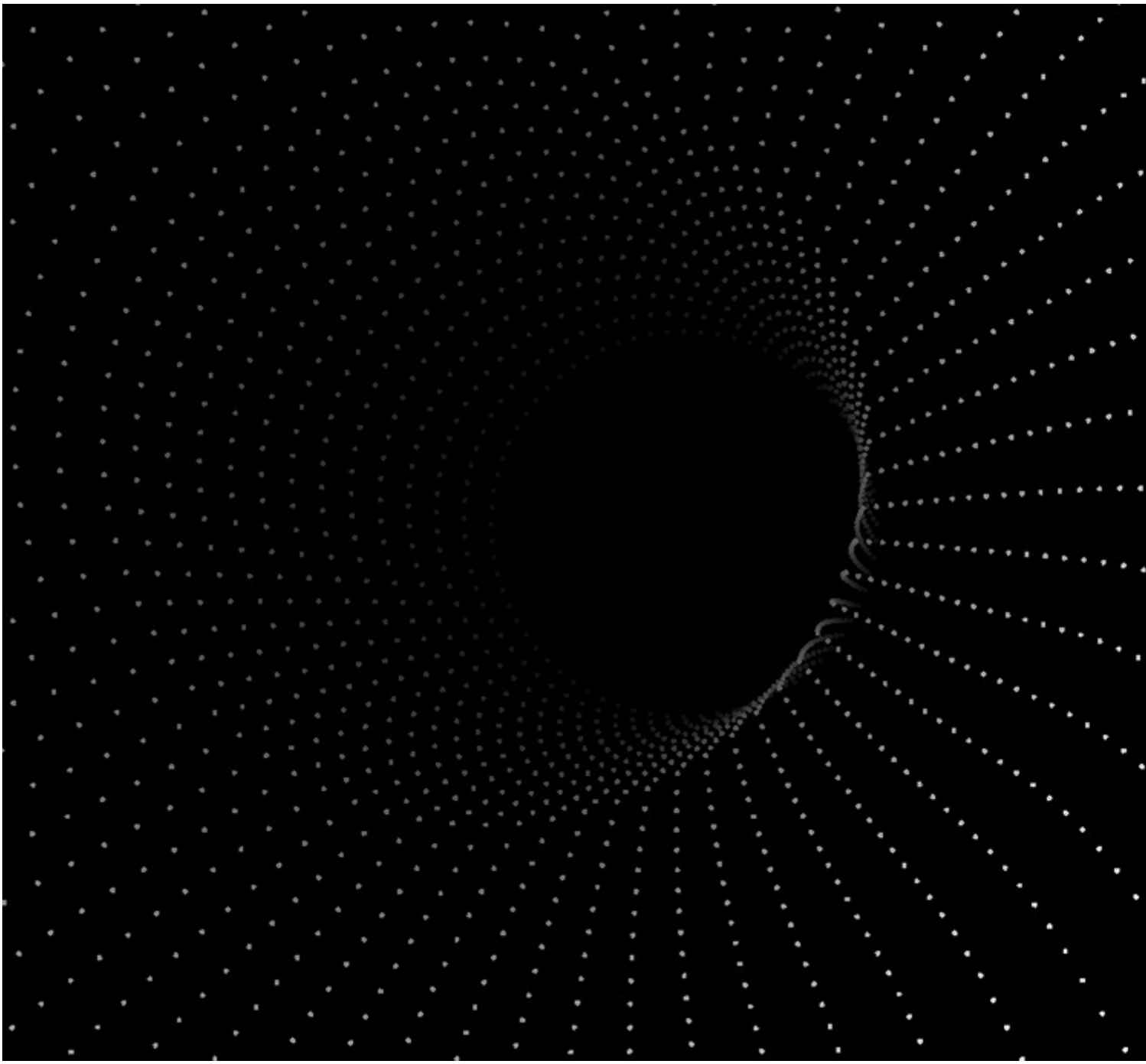
Die drei Zeilen darunter sind einfacher. Erst sagen wir dem erstellten Objekt, in welche Richtung er fliegen soll. Die zweite Zeile könnte man auch löschen, da *xx += 0;* nichts bewirkt. Sie ist ein Platzhalter. Du kannst ja mal schauen, was passiert, wenn Du aus der 0 eine 5 machst. In der dritten Zeile setzen wir die Richtung um 5 hoch. Da wir das 72Mal tun (Schleife!) erhalten wir somit einen geschlossenen Kreis. Auch hier kann man mit dem Wert experimentieren. Der Tunnel sieht zum Beispiel auch ganz interessant aus, wenn man statt 5 die Zahl 2 nimmt.

Im nächsten Abschnitt verändern wir den Radius. Er wird laufend um 4 Pixel erhöht, bis er die 190 erreicht hat, dann wird er verkleinert, bis er bei 80 angekommen ist. Ab dann vergrößert sich der Radius wieder bis 190 und so weiter.

Am Ende angekommen wird der Alarm neu gestartet, und zwar nach 0.05 Sekunden. Das sind, bei einer Raumgeschwindigkeit von 60 Frames, genau 3 Steps. Wer etwas mehr Pixel auf dem Bildschirm haben möchte, kann den Wert runter setzen, aber kleiner 0.0166 lohnt sich nicht und dann kann man das Ganze auch in den **Step-Event** verschieben. Wer weniger Pixel möchte, kann es Mal mit 0.15 versuchen.

Das war es auch schon. Wenn Du möchtest, kannst Du noch einen **release<Escape>-Event** erstellen und dort als Code *game_end();* schreiben. Dann kannst Du den Effekt mit der Escape-Taste abbrechen.

Der fertige Effekt sieht dann so aus:



Pixeltunnel-Effekt

1 Datei(en) 194.59 KB

Download

Date Created
19. Oktober 2016
Author
sven