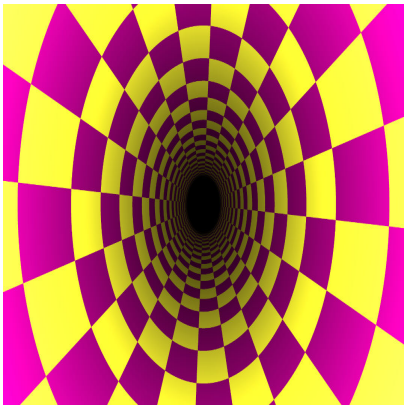

Shader-Effekt: Tunnel mit Maussteuerung

Description

Im dritten Teil der Shader-Serie wurde bereits ein Spiraltunnel gezeigt. Heute geht es im einen klassischen Tunneleffekt, der recht flexibel angepasst werden kann. Außerdem lassen sich x- und y-Koordinaten mit der Maus beeinflussen. Das macht es als Einsatz für Spiele interessant.

Hier ein kurzes Video, in dem zu sehen ist, was wir vorhaben:

https://www.bytegame.de/wp-content/uploads/2024/01/Tunneleffekt.mp4

Zu sehen ist ein klassischer Tunneleffekt. Der Tunnel besteht aus einem hübschen Schachbrettmuster, wir können die Position per Maus verändern und am Ende des Tunnels sehen wir einen schwarzen Kreis.

Wozu ist das gut?

Wie so viele Effekte, die in diversen Tutorials gezeigt wurden, kann man auch diesen Effekt für Menüs oder Credits verwenden. Da wir die Position per Maus verändern können, aber auch ganz generell, eignet es sich sogar als Hintergrund für ein Spiel. Man könnte die Maussteuerung blockieren und stattdessen ein Raumschiff steuern, welches durch den Tunnel fliegt. Ihm kommen feindliche Raumschiffe entgegen, die man abschießen kann. Oder Gesteinstrümmer, denen man ausweichen muss.

Objekt vorbereiten

Theoretisch kann man, wenn man will, die Maussteuerung weglassen. Ich zeige es mit eingebauter Maus, löschen kann man immer noch.

Mein Objekt heißt `o_tube_01`. Im **Create-Event** weichen wir von der üblichen Vorgehensweise ein wenig ab:

```
time = 0;
time_add = 0.02;
tubeSpeed = 0.2;

window_mouse_set(window_get_width()/2, window_get_height()/2);
```

Die ersten beiden Zeilen kennen wir bereits aus zahlreichen Tutorials. `tubeSpeed = 0.2` ist die Geschwindigkeit des Tunnels. Damit man im Spiel flexibel ist, ist es wichtig, diese Variable in GML statt im Shader zu beeinflussen.

Die letzte Zeile setzt den Mauszeiger auf die Mitte des Bildschirms. Somit stellen wir sicher, dass der Tunnel immer in der gleichen Position beginnt.

Im **Draw-Event** erwarten uns keine Überraschungen. Hier geben wir, im Vergleich zu anderen Tutorials, lediglich noch die Mausposition und die Geschwindigkeit weiter:

```
time += time_add;

shader_set(sh_tube_01);
shader_set_uniform_f(shader_get_uniform(sh_tube_01,"mouse"), mouse_x, mouse_y)
shader_set_uniform_f(shader_get_uniform(sh_tube_01,"speed"), tubeSpeed);
shader_set_uniform_f(shader_get_uniform(sh_tube_01,"resolution"), display_get_
shader_set_uniform_f(shader_get_uniform(sh_tube_01,"time"), time);
draw_surface_ext(application_surface, 0, 0, 1, 1, 0, c_white, 0);
shader_reset();
```

Wie man sehen kann, heißt der Shader bei mir `sh_tube_01`.

sh_tube_01

Wir nutzen nur den Fractal-Shader. Hier ist der kommentierte Code:

```
// Uniform-Variablen, die von außen gesetzt werden können
uniform float time;         // Zeitvariable
uniform float speed;        // Geschwindigkeitsvariable
uniform vec2 mouse;         // Mausposition
uniform vec2 resolution;    // Bildschirmauflösung

// Empfindlichkeit der Mausbewegung
#define sensitivity 0.005

// Funktion zur Berechnung des Schachbrettmusters
float getCuteCheckerboard(vec2 position) {
    float xpos = floor(40.0 * position.x);
    float ypos = floor(20.0 * position.y);
    float col = mod(xpos + ypos, 2.0);
    return col;
}

// Hauptshader-Funktion
void main(void) {
    // Normalisierte Bildschirmkoordinaten von -1 bis 1 mit (0, 0) in der Mitt
```

```
    vec2 position = 2.0 * (gl_FragCoord.xy - 0.5 * resolution.xy) / resolution

    // Anpassung der Position basierend auf Mausbewegung
    position.x += 2.0 / resolution.x / resolution.y + ((mouse.x - resolution.x
    position.y += 2.0 / resolution.x / resolution.y + ((mouse.y - resolution.y

    // Umwandlung von kartesischen Koordinaten in polar
    float a = atan(position.y, position.x);
    float r = length(position);

    // UV-Koordinaten für Texturabtastung
    vec2 uv = vec2(0.1 / r + speed * time, (a + speed * time) / (2.0 * 3.1416)

    // Schachbrettmuster abrufen
    float checkerboard = getCuteCheckerboard(uv);

    // Definierte Farben für das Schachbrettmuster
    vec3 color1 = vec3(0.568, 0.004, 0.431);  // #91006e
    vec3 color2 = vec3(1.000, 0.976, 0.192);  // #fff831

    // Mischen der Farben basierend auf dem Schachbrettmuster und Dunkelheit a
    vec3 finalColor = mix(color1, color2, checkerboard) * smoothstep(0.8, 1.0,

    // Farbausgabe des Pixels
    gl_FragColor = vec4(finalColor * r, 1.0);
}
```

Erklärungen

Ganz oben holen wir uns die Variablen von `o_tube_01`. Außerdem definieren wir die Variable `sensitivity`. Das bezieht sich auf die Maus. Je größer der Wert ist, umso sensibler reagiert der Tunnel auf die Mausbewegung. Eventuell wäre es sinnvoll, diesen Wert an die Auflösung anzupassen, auch wenn Tests zeigten, dass `0.005` bei Auflösungen zwischen 320×200 und 3840×2160 ganz gut funktioniert.

Die Funktion `getCuteCheckerboard` berechnet ein Schachbrettmuster, also die Textur des Tunnels.

`float xpos = floor(40.0 * position.x);`: Hier wird die horizontale Position `xpos` berechnet, indem die x-Komponente der Eingabeposition `position.x` mit 40 multipliziert und auf die nächste ganze Zahl abgerundet wird.

`float ypos = floor(20.0 * position.y);`: Ähnlich wie bei `xpos` wird die vertikale Position `ypos` berechnet. Die y-Komponente der Eingabeposition `position.y` wird mit 20 multipliziert und auf die nächste ganze Zahl abgerundet.

`float col = mod(xpos + ypos, 2.0);`: Die `mod`-Funktion wird verwendet, um den Rest der Division von `xpos + ypos` durch 2 zu berechnen. Dadurch entsteht ein Schachbrettmuster mit abwechselnden 0 und 1, da gerade Zahlen den Rest 0 haben und ungerade Zahlen den Rest 1.

`return col;`: Die berechnete Farbe der Textur wird zurückgegeben. Diese Funktion gibt 0 für schwarze Flächen und 1 für weiße Flächen des Schachbrettmusters zurück. Eingefärbt wird es dann

später in der Hauptfunktion.

Hauptfunktion

Die Hauptfunktion `main` des Shaders ist für die Berechnung der Farbe eines jeden Pixels auf dem Bildschirm verantwortlich.

`vec2 position = 2.0 * (gl_FragCoord.xy - 0.5 * resolution.xy) / resolution.y;`: Normalisierte Bildschirmkoordinaten werden berechnet, wobei der Ursprung (0,0) in der Mitte liegt. Diese Koordinaten werden dann durch die Bildschirmauflösung geteilt, um Werte im Bereich von -1 bis 1 zu erhalten.

`position.x += 2.0 / resolution.x / resolution.y + ((mouse.x - resolution.x / 2.0) * sensitivity);`: Die Position wird basierend auf der Mausbewegung angepasst. Der Mausinput beeinflusst die horizontale Position `position.x`, wodurch eine Art Steuerung des visuellen Effekts ermöglicht wird.

`position.y += 2.0 / resolution.x / resolution.y + ((mouse.y - resolution.y / 2.0) * sensitivity);`: Ähnlich wie bei der horizontalen Position wird die vertikale Position `position.y` durch die Mausbewegung beeinflusst.

`float a = atan(position.y, position.x);`: Die kartesischen Koordinaten werden in polare Koordinaten umgewandelt. a repräsentiert den Winkel (Arkustangens), der für die Berechnung der Polar-Koordinaten benötigt wird.

`float r = length(position);`: Der Abstand vom Ursprung wird berechnet, um die Distanz des Pixels vom Zentrum zu erhalten.

`vec2 uv = vec2(0.1 / r + speed * time, (a + speed * time) / (2.0 * 3.1416));` : UV-Koordinaten werden berechnet, um die Textur zu sampeln. Der Abstand `0.1 / r` sorgt für eine Art Zoom-Effekt basierend auf der Entfernung zum Zentrum.

`float checkerboard = getCuteCheckerboard(uv);`: Die Funktion `getCuteCheckerboard` wird aufgerufen, um ein Schachbrettmuster zu generieren.

`vec3 color1 = vec3(0.568, 0.004, 0.431);`: Zwei Farben (`color1` und `color2`) werden definiert, die später gemischt werden, um die endgültige Farbe des Pixels zu erzeugen.

`vec3 finalColor = mix(color1, color2, checkerboard) * smoothstep(0.8, 1.0, r * 6.0);`: Die Farben werden basierend auf dem Schachbrettmuster gemischt. Der Dunkelheitseffekt am Ende des Tunnels wird durch `smoothstep` hinzugefügt. Somit sieht man nicht, wie der Effekt am Ende entsteht und der Tunnel wirkt „unendlich”.

`gl_FragColor = vec4(finalColor * r, 1.0);`: Die endgültige Farbe des Pixels wird mit der Entfernung `r` vom Zentrum multipliziert und als Ausgabe für den Pixel gesetzt.

Hintergrundwissen

Kartesische Koordinaten

Kartesische Koordinaten sind eine Möglichkeit, die Position eines Punktes im Raum oder auf einer Ebene zu beschreiben. Ein Punkt in kartesischen Koordinaten wird durch zwei oder drei Werte dargestellt: `(x, y)` für die Ebene oder `(x, y, z)` im Raum. Die x-Koordinate repräsentiert die horizontale Position, die y-Koordinate die vertikale Position und bei dreidimensionalen Koordinaten repräsentiert die z-Koordinate die Tiefe.

Arkustangens (Arctan oder atan)

Der Arkustangens ist eine mathematische Funktion, die das Verhältnis zweier Seiten eines rechtwinkligen Dreiecks berechnet. Genauer gesagt, gibt der Arkustangens das Winkelmaß (in Radianten) zurück, dessen Tangens das Verhältnis von Gegenkathete zu Ankathete ist.

Angenommen, wir haben ein rechtwinkliges Dreieck, in dem die Länge der Gegenkathete 3 und die Länge der Ankathete 4 beträgt. Wir möchten den Winkel ? (Theta) berechnen, der dem Verhältnis von Gegenkathete zu Ankathete entspricht.

Das Verhältnis von Gegenkathete zu Ankathete ist gegeben durch den Tangens des Winkels ?:

$$tan(\theta) = \frac{Gegenkathete}{Ankathete}$$

In diesem Beispiel ist tan(?)=¾?.

Um nun den Winkel ? zu berechnen, verwenden wir den Arkustangens (`atan`):

$$\theta = tan^{-1}(\frac{3}{4})$$

Wenn wir dies berechnen, erhalten wir den Winkel in Radianten. Wir können den Winkel auch in Grad umrechnen, wenn gewünscht.

Nun berechnen wir dies:

$$\theta = tan^{-1}(\frac{3}{4}) \approx 0.6435 Radianten$$

Das wäre der Winkel ? für das gegebene rechtwinklige Dreieck mit einer Gegenkathete von 3 und einer Ankathete von 4.

Solltest du auf deinem Taschenrechner nachrechnen und eine Zahl wie 36,8698977 (grob gerundet) erhalten, liegt es daran, dass der Rechner gleich in Grad statt in Radianten anzeigt. Die Umrechnung

lautet:

$$Grad = Radianten \cdot \frac{180}{\pi}$$

Wieso wird der Arkustangens für die Umwandlung verwendet?

Wenn man von kartesischen Koordinaten zu polaren Koordinaten wechselt (z. B. wenn man die Position eines Punktes im Raum in Entfernung und Winkel darstellen möchte), benötigt man eine Möglichkeit, den Winkel zu berechnen. Der Arkustangens erfüllt genau diese Funktion, indem er das Verhältnis von y zu x verwendet.

In polaren Koordinaten repräsentiert der Radius (`r`) die Entfernung vom Ursprung, und der Winkel (? oder a im Code) gibt die Richtung an. Die Umwandlung von kartesischen zu polaren Koordinaten ist also der Prozess, die x- und y-Koordinaten eines Punktes in die Entfernung (`r`) und den Winkel (a) umzuwandeln. Dadurch entsteht der räumliche Effekt.

Übrigens: Wenn man Raum und View auf 320×200 stellt, sieht es richtig schön Retro aus. Wie in den alten DOS-Zeiten:

https://www.bytegame.de/wp-content/uploads/2024/01/Tunneleffekt2.mp4

Hier noch ein Beispiel, wo dieser Effekt – und viele weitere – verwendet wurde. In der Demo wird allerdings kein Schachbrettmuster, sondern eine rostige Textur benutzt:

Weiterführende Links

Shader-Programmierung 1: Grundlagen und Sprites
Shader-Effekt: Warping
Shader-Effekt: Interferenz-Effekt

Date Created
10. Januar 2024
Author
sven