Rob3rt

Team

Stefan Zaufl

~~Martina Segnstschmid~~(ehemals, hat LVA abgebrochen)

Steuerung

WASD – Charakter bewegen

Maus – Kamera drehen

Space – Springen

E – Interagieren

T – Zeit wechseln

ESC – Beenden

Projektstatus

Zur Zeit ist das Spiel noch im "Alpha"-Status: Es gibt schon ein Spielziel, das Basis-Gameplay wurde realisiert, doch einige zusätzliche Features und Content konnten leider aufgrund von mangelnden Ressourcen nicht rechtzeitig fertiggestellt werden.

Features

- Level werden aus Textfiles generiert(siehe bin/levels/LevelFileFormat.TXT für nähere Informationen)
- Directional Shadow-Mapping(Am Besten zu sehen bei der Zeitmaschine oder bei Durchgängen, Türen)
- Zeitreisen
- Prozedurale Textur sorgt beim Zeitreisen für den „Waber-Effekt“
- Wenn Interaktionen mit der Spielwelt möglich sind, werden Bildschirm-Hinweise eingeblendet.
    - Türen können geöffnet werden, wenn sie nicht verschlossen sind
    - Die Zeitmaschine kann aktiviert werden, um ins nächste Level zu kommen
- Animation der Zeitmaschine, wenn diese aktiviert wird.

Effekte

- Directional Shadow-Mapping
    - http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
- Prozedurale Texturen
    - http://en.wikipedia.org/wiki/Procedural_texture
    - https://github.com/ashima/webgl-noise/

Lichtquellen

Die Beleuchtung geht von einer Stirnlampe des Spielers aus. Alle im Spiel befindlichen Objekte werden von dieser einen Lichtquelle beleuchtet. Es wird Lambert-Phong-Shading dafür benutzt.

Modelle

Alle Modelle wurden von David Portisch, einem Freund von mir, erstellt und mir zur Verfügung gestellt. Er hat zum Erstellen der Modelle Blender(http://www.blender.org/) benutzt. Simple Modelle wie Planes, Spheres oder Würfel habe ich selbst in Blender erstellt.

Musik

Die Musik in dem Spiel wurde von Martin Fritsch(marDin Fridge), einem Freund von mir, komponiert und unter der CC By Lizenz veröffentlicht(siehe bin/sound/legal.txt). Die 2 Musikstücke sind so aufeinander abgestimmt, dass man einen Übergang zwischen den beiden nicht bemerkt.

Details zur Implementierung

Shadow Mapping

Beim Shadow mapping rendert man die Szene 2 mal: Das 1. Mal aus Sicht der Lichtquelle, dann aus Sicht der Kamera. Dann ordnet man jeden möglichen Pixel des Kamera-Bildes einen Pixel des Licht-Bildes zu und vergleicht die Tiefeninformationen. Ist der Pixel gleich oder weniger weit von der Lichtquelle entfernt, wie der Pixel auf dem Lichtquellen-Bild, ist der Pixel beleuchtet, ansonsten im Schatten. Um Shadow-Acne zu reduzieren wurden alle Pixel im Licht-Bild um einen Fixen Betrag nach hinten verschoben. Damit es eine gerichtete Lichtquelle ist, wurde eine Projektionsmatrix anstelle einer Orthogonalen- als View-Matrix verwendet. Die Position auf der Shadowmap wird mehrfach ausgelesen, da für den Texture-Lookup nicht der sampler2D, sonder der sampler2Dshadow verwendet wurde.

Prozedurale Texturen

Für jeden Pixel des Bildes wird eine Funktion verwendet, die pseudo- zufällig unscharfe Kreise generiert. Man kann sich das alleine schon als ein Bild vorstellen, nur dass es durch eine Mathematische (3D)Funktion gegeben ist. Für diese Funktion verwende ich snoise(Link siehe Effekte), da die in openGL integrierte noise-Funktion immer den Wert 0,5 zurück geliefert hat. Als seed fr snoise habe ich die normalisierten x und y Koordinaten des Pixels herangezogen, der z Wert wird von der Zeit bestimmt, dadurch bewegt sich die 3-Dimensionale Textur durch die Bildfläche und es kommt zum Wabern.

Nun macht man mehrere Lookups auf dieser Prozeduralen Textur mit unterschiedlichen Zoom-Stufen und mittelt die Ergebnisse. So bekommt man eine viel detailreichere Textur zurück, die sich in sich wiederholt. Danach muss dieser Grauwert noch einer Farbe zugeordnet werden. Dies ist so gelößt, dass fast alle Werte auf schwarz gemappt werden, nur 2 blaue schmale Bänder um 0,5 herum bilden die Ausnahme.

View-Frustum-Culling

Das View-Frustum-Culling findet ausschließlich in Weltkoordinaten statt. Zuerst werden alle benötigten Informationen gesammelt: Dimensionen der near- und farplane, berechnet aus FOV, aspect & near- und fardistance. Dann werden die Kameraposition, Blickrichtung, Oben- und Rechts- Vektor der Kamera mithilfe der Kameramatrix berechnet. Dann wird nur noch der linke

untere nahe Punkt und der rechte obere weite Punkt berechnet und die 4 Vektoren, die am Sichtfeld entlang gehen(von der Kameraposition nach links, rechts, oben und unten). Diese 4 Vektoren werden als Operand des Kreuzprodukts mit den Oben & Rechtsvektoren der Kamera benutzt, um die Normalen der Viewplanes zu extrahieren. Die Normalen der far- und nearplane sind die inverse Blickrichtung bzw. die Blickrichtung. Mit den 2 vorher berechneten Punkten können nun die 6 View-Planes aufgestellt werden. Danach wird für jedes Objekt geprüft, ob es sich innerhalb aller 6 View-Planes befindet(vorzeichenbehafteter Abstand zur Plane muss > 0 - Radius des Objekts sein).

Tutorial: http://www.lighthouse3d.com/tutorials/view-frustum-culling/

Animationen

Es wurden nur hierarchische Animationen implementiert. Das zu bewegende Objekt holt sich die Model-Matrix seines Parents und multipliziert diese mit seiner eigenen. Die Animationen selbst wurden hartgecoded(wenn timer > 2, dann bewege Objekt nach +x, …).

Komplettlösung

Hier eine Schritt-für-Schritt Anleitung, um das Spiel durchzuspielen.

Level 1

Zur Tür gehen, Tür öffnen(E), in den nächsten Raum zur Zeitmaschine gehen und vor der Zeitmaschine auf E drücken.(Ein 2. Drücken von E vor der Zeitmaschine bricht die Animation ab)

Level 2

Zeit wechseln(T), aus dem Becken steigen und die große Treppe nehmen(an einer Wände). Die Tür am Ende der Treppe ist verschlossen, darum noch einmal Zeit wechseln(T), jetzt liegt die Tür am Boden, zur Zeitmaschine gehen und E drücken. Spiel geschafft!

Additional Libraries

- boost (Implementierungen von häufig gebrauchten Features in C++) - http://www.boost.org/
- bullet(Kollisions-Library) - http://bulletphysics.org/wordpress/
- FreeImage(Bilder lade/speichern Library) - http://freeimage.sourceforge.net/
- irr-Klang(Sound-Library) - http://www.ambiera.com/irrklang/
- glew(Laden von OpenGL-Extensions) - http://glew.sourceforge.net/
- glfw(Öffnen & Verwalten von einem Fenster) - http://www.glfw.org/
- glm(Mathematik-Library für OpenGL) - http://glm.g-truc.net/