OBLIVIA
Ein dystopisch dunkles Survival-Spiel

00271430 Lisalotte Tscherteu
11701221 Tobias Batik

Oblivia ist ein singleplayer Survival-Spiel. Das Ziel des/der Spielers_In ist es möglich lange nicht in ein hell Strahlendes Partikel System zu laufen.

Gameplay
Der/die Spieler_In steuert sich mit der Tastatur durch ein dystopisches dunkles dreidimensionales Labyrinth. Der/die Spieler_In soll zunehmend die Orientierung im Gängesystem verlieren. Während des Irren durch das Labyrinth, begegnet man hell leuchtenden Partikel systemen denen man ausweichen oder vor ihnen fliehen muss. Wenn man zu nah an die Partikel kommt stirbt man. Das Partikel System breitet sich im Raum immer weiter aus bis es erlischt.

Die Spielwelt ist in drei sektoren eingeteilt. Je Nachdem in welchem Bereich der Welt man sich befindet werden zufällig neue Particle Systeme generiert.

Die Spielwelt wurde mit dem Onlinetool threejs editor gebaut. Anschließend wurden die Koordinaten und dimensionen der Objekte in die Szene übertragen.

übersicht Spielewelt

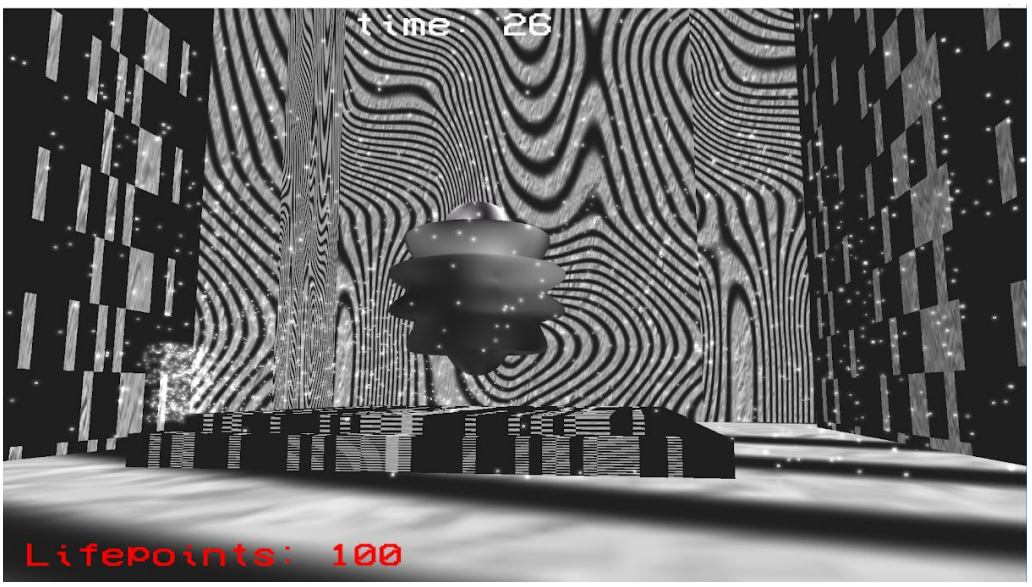
https://threejs.org/editor/

Controls
Das Spiel wird in der First Person perspektive gespielt. Der Spieler kann sich mithilfe der Tastatur Pfeiltasten oder den Keys W,A,S,D durch die Szene Bewegen. Mithilfe der Pfeiltaste Links, Rechts beziehungsweise A und D kann die Kamera um die Eigene Achse rotiert werden. Wie weit sich die Kamera pro Frame bewegt ist Abhängig von der Framerate. Mithilfe von F8 kann die Steuerung auch über die Maus erfolgen.

Win / Loose
Der Spieler hat am Anfang des Spieles 100 Lebenspunkte. Wenn er von einem “Tödlichen” Partikel System getroffen wird, verringern sich seine Lebenspunkte. Sobald die Lebenspunkte <= 0 sind, hat der/die Spieler_in das Spiel verloren. Das Spiel ist gewonnen, wenn der/die Spieler_in länger als eine bestimmte zeit am Leben bleibt (60 Sekunden).

Moving Objects / Animation
Im Spiel sind einerseits die Particles animiert, Prozeduralen Texturen, Spheres mithilfe einer Vertex Shader animation sowie das direct light ist animiert. Eine Sphere, zu der man mit F3 kommt, bewegt sich zusätzlich im Raum auf und ab.

Lights
Directional Light
Die Farbe des Lichtes ist animiert und verändert sich im laufe der Zeit (wechselt auf rot).



Point Lights
Es gibt im Spiel vier Point Lights. Die Lichter sind jeweils über bzw unter einer der deformierten Sphere positioniert, sowie eines beim, durch Assimp geladenes Model. Die Farbe der Point Lights ist Lila.

Modelloader
Mithilfe von Assimp können modelle geladen werden und in der Model class gespeichert bzw. verwendet werden. Für das Spiel wurde das geladen Objekt nicht selbst erstellt. Sondern aus dem Internet geladen.

https://www.youtube.com/watch?v=W_Ey_YPUjMk
http://www.assimp.org/
https://learnopengl.com/Model-Loading/Assimp

Settings File - settings.ini

width, height	Auflösung: Getestet 1280 x 720, 1920 x 1080 Wenn das Spiel im Fullscreen Modus gestartet wird, wird die auflösung des Bildschirms verwendet
refresh_rate	refresh rate
fullscreen	Fullscreen true / false Wenn true, wird die auflösung des Bildschirm für das Spiel verwendet.
brightness	default 2.0f, damit kann die Helligkeit des Spieles manimuliert werden
MaxPartperSystem	Anzahl der Particles pro System. Manche Systeme erzeugen doppelt so viele Particles. Getestet bis 8000.
avgParticleLifetime	default 1.0f, damit kann die lebenszeit der Particles verändert werden.
gameLength	default 60.0f, gibt an wie viele Sekunden der Spieler überleben muss bis er gewinnt.
difficulty	default 1.0f, damit kann die schwierigkeit des Spiels gesteigert werden. Umso niedriger der Wert umso mehr Particle Systeme werden generiert und damit umso schwerer ist das Spiel

Keys (für Debugging)

F1 Wireframe Darstellung, funktioniert nur wenn der Bloom Effekt ausgestellt ist.
F2 Backface culling toggl.
F3 Spieler “fliegt” zu einer Moving Sphere.
F4 Blendet das Headsup display ein / aus.
F5 Schalten den Bloom effekt ein / aus.
F6 Können die Wände Ausgeblendet werden.
F7 Schaltet Normal Mapping ein / aus.
F8 Kamera folgt der Maus ein / aus.

PhysX

Ein Cameracontroller, CapsuleController, kontrolliert die Collisions Detection und Resolution zwischen den Objekten und dem Spieler. Verwendet wird PhysX 4.1.

https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html
https://github.com/NVIDIAGameWorks/PhysX

Heads up Display

Das HUD zeigt die verbleibende Zeit und "Lifepoints".
Mit F4 kann man die Sichtbarkeit des HUD toggeln.
Das HUD verwendet die Freetyp-Library.

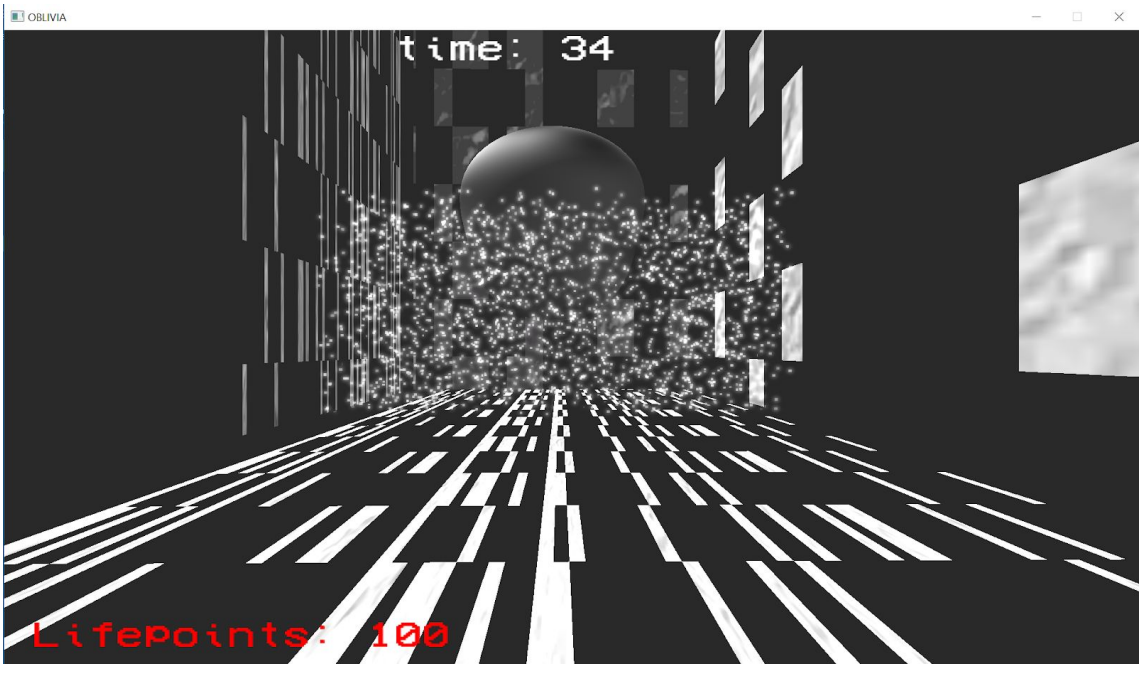
https://www.freetype.org/
https://learnopengl.com/In-Practice/2D-Game/Render-text

Effekte

CPU-Partikel-System
Die Position der Einzelnen Partikels wird in der Klasse ‘partical’ und ‘Partsystem’ für jedes Frame aktualisiert.
Das Partikel System hat bei der Erstellung eine vorgegebene größe (6000 Partikel). Nicht alle Partikel haben eine gleich lange Lebensdauer (random), daher werden in jedem Frame einzelne partikel aus dem System entfernt. Die Particle Systeme werden, an 16 Verschiedenen Positionen erzeugt. Je nachdem in welchem Bereich sich der/die Spieler_in befindet werden die Particle Systeme erzeugt. Im schnitt sind ungefähr 3-4 Particle System gleichzeitig aktiv. Im Settings file kann mit hilfe der variable “MaxPartperSystem” die anzahl der Particles, bei der erzeugung angegeben. Einige Particle Systeme generieren die doppelte Menge an Particles. Auf unseren Rechnen haben wir es bis 8000 getestet, was ohne Probleme funktioniert hat.

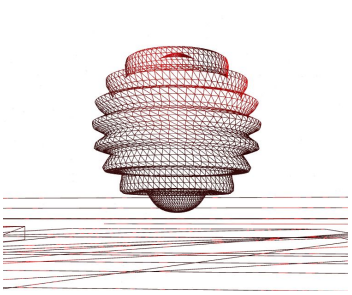
http://genericgamedev.com/effects/cpu-vs-gpu-particles-from-20-to-200-fps/
http://genericgamedev.com/effects/parametric-gpu-accelerated-particles/
https://learnopengl.com/In-Practice/2D-Game/Particles

Bloom
Vom Bloom oder Glow sind nur die Particles betroffen.
Der Particle Shader schreibt in ein zweites Color Attachment. Die anderen Shader schreiben nur in das erste Attachment des selben Framebuffers. Das particle-color-attachment wird mit einem Gauss-Filter geblurrt und anschließend über die "normal" gerenderte Szene gelegt.

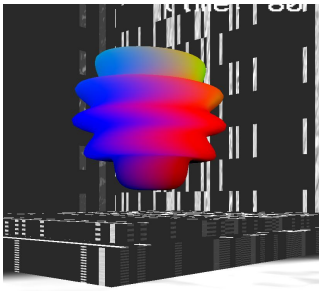


https://en.wikipedia.org/wiki/Bloom_(shader_effect)
https://learnopengl.com/Advanced-Lighting/Bloom
https://learnopengl.com/Advanced-OpenGL/Framebuffers

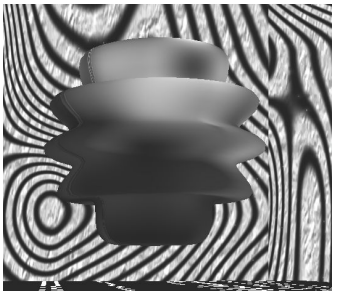
Vertex Shader animation
Mithilfe des Vertexsahder (vertAnimation.vert, vertAnimation2.vert, vertAnimation3.vert) werden die Vertices einer Sphere verschoben. Der Radius wird mithilfe von sin und cos funktionen verändert. Dem shader wird bei jedem frame, die Zeit übergeben. Jeder Shader deformiert die Sphere, auf die der Shader angewendet wird, unterschiedlich. Die Vertex Shader animation kommt im Spiel in kombination mit einer prozeduralen Noise Texture zum einsatz.



wireframe



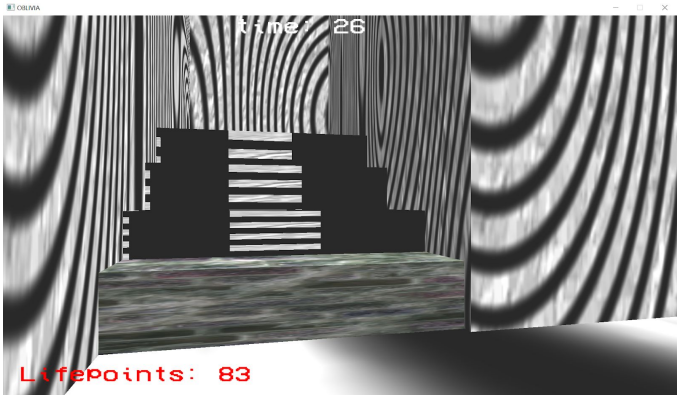
deformierte Sphere mit Normal Material



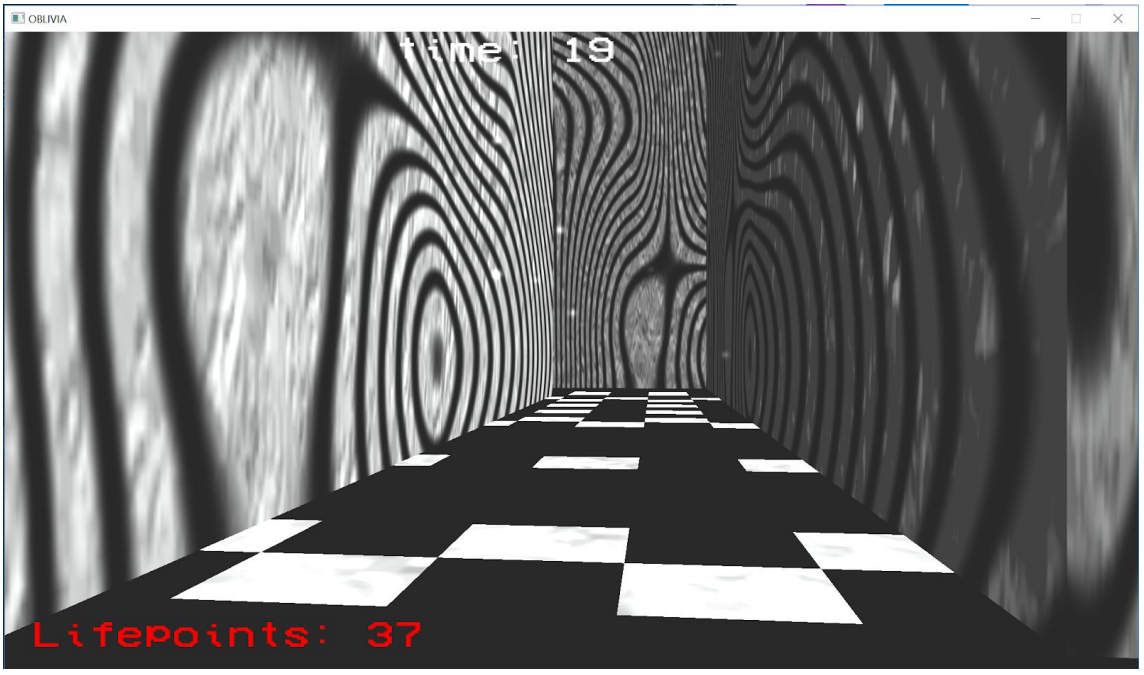
deformierte Sphere mit einer prozeduralen noise texture

https://blog.mozvr.com/water-ripples-with-vertex-shaders/

Textures
Aus design technischen Gründen werden nur sehr wenige nicht generierte Texturen verwendet.



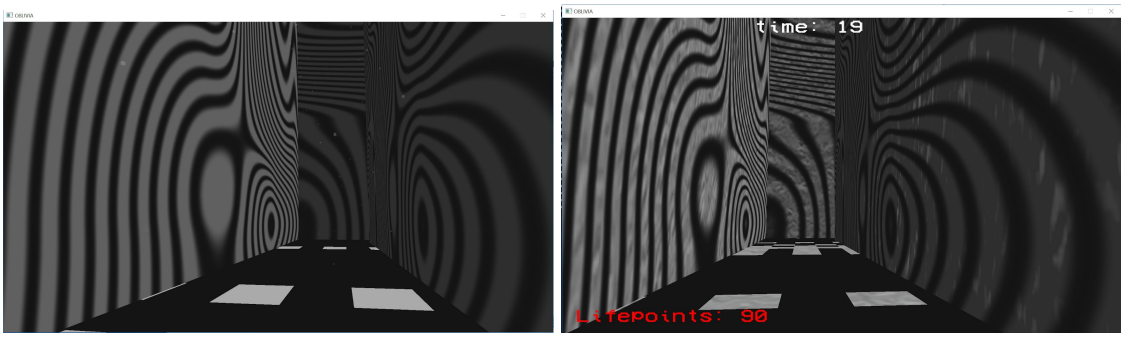
Prozedurale Texturen
Es sind 3 verschiedene Texturen im Einsatz, sie verändern sich in Abhängigkeit der Zeit.
Die einer Holzmaserung (proced3.frag) und Wolken (proced2.frag) ähnlichen Texturen basieren auf einer 2D Noise Funktion.
Die zweite Texture (proced2.frag) basiert auf einer 2D Random Funktion.
Proced1 und Proced3 wird auf den Wänden und Böden des Labyrinths verwendet, Proced2 auf den Spheres.
https://thebookofshaders.com/11/
https://thebookofshaders.com/10/



https://thebookofshaders.com/11/
https://en.wikipedia.org/wiki/Simplex_noise
http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf

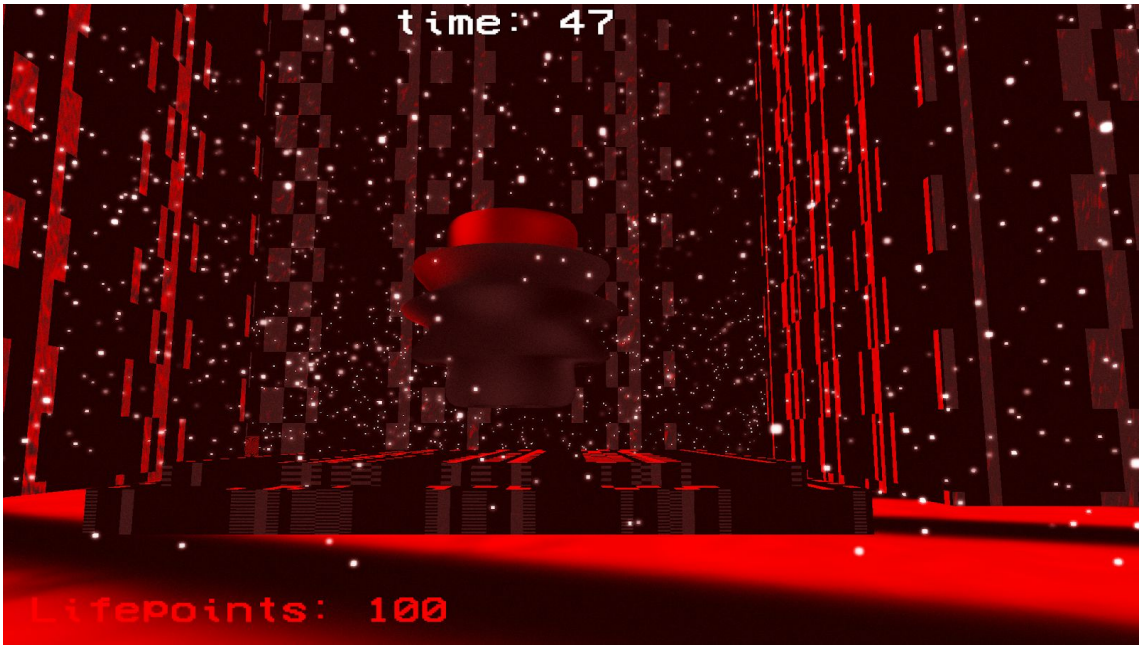
Normal Map

In den Shadern proced1.frag, proced2.frag, proced3.frag ist einfach Normal mapping implementiert. Es wird der neue Normal Vector aus der Textur "normalMap" gelesen, und für die jeweilige Seite eines Cubes transformiert. Mithilfe von F7 kann normal mapping ein und ausgeschalten werden. Normalmapping spielt eine große rolle in der Ästhetik des Spieles und kommt auf den Wänden und dem Boden zum einsatz.



kein Normal Mapping *Normal Mapping*

https://en.wikipedia.org/wiki/Normal_mapping
https://learnopengl.com/Advanced-Lighting/Normal-Mapping



00271430 Lisalotte Tscherteu 11701221 Tobias Batik