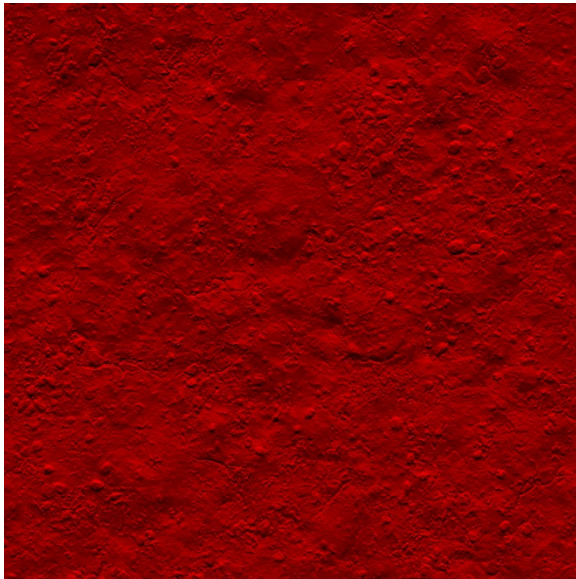


Description of Implementation

Gameplay

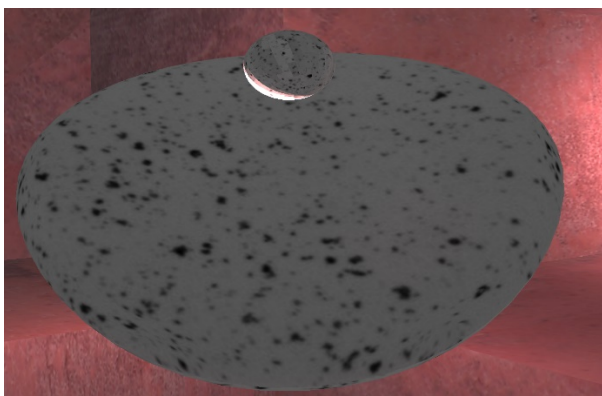
Ziel des Spiels ist es die verwundbaren Wände, die im Level verstreut sind mit der *Space*-Taste anzugreifen und zu zerstören. Diese Wände haben folgende Textur:



Dabei muss man Nahe genug sein, um Schaden zuzufügen. Dabei entweichen rote Partikeln, die aus der Mitte der Wand entgleiten und die Textur verschwindet langsam, wenn der Angriff durchgeführt wird.



Im Spiel gibt es als Gegner folgende:

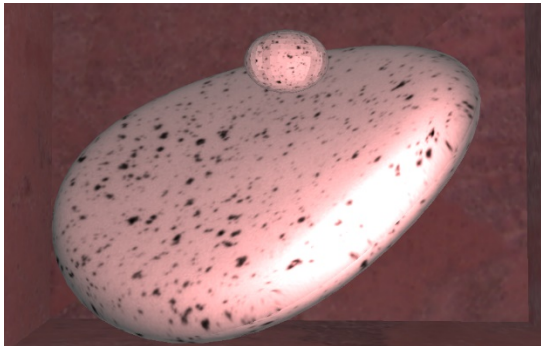


Wenn man diese berührt, verliert man langsam Leben. Wenn man nur noch 0 Leben hat, hat man verloren und beginnt von neu.

Die Level sind so designed, dass sie vertikal in die Höhe oder Tiefe gehen. Dadurch wird die 3 Dimension ideal ausgenutzt. Außerdem darf man an verwundbaren Wänden und Wänden mit prozeduralen Texturen nicht ankommen, da man auch dort Leben verliert. Man muss also nahe genug an einer verwundbaren Wand sein, aber nicht zu nahe.

Complex 3D Models from Files

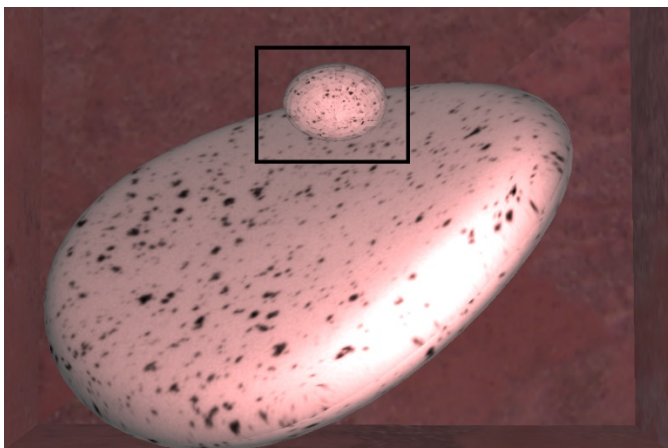
Als komplexes Model wird diese Zelle genutzt.



Diese nicht konvex

Hierarchical Animation Using Physics Engine

Die hierarchische Animation wird dadurch erzielt das eine Kugel über der Zelle rotiert.



Diese rotiert sich um die horizontale Achse und bewegt sich auch mit der Zelle mit.

View-Frustum Culling

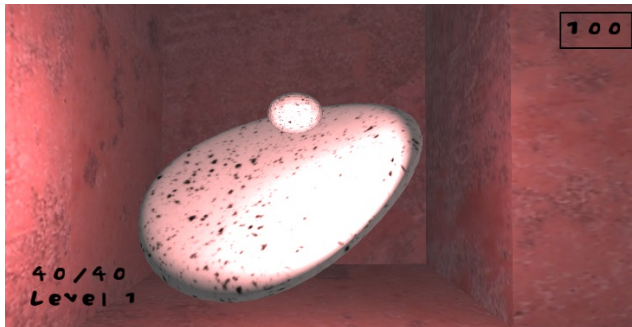
Aus den Kamera – Werten werden 6 Planes, die deine Sichtgrenze darstellt, erstellt. Für jedes Objekt wird ein Schwerpunkt und von dem die Distanz zur weitersten Vertice errechnet, welche dann dein Radius darstellt. Dadurch wird jedes Objekt für das View - Frustum als Sphere behandelt und schließlich ermittelt, ob das Objekt ganz oder teilweise im View – Frustum zu sehen ist.

Debug Options

F1 - es gibt keine Hilfestellung

F2 – Frametime on/off

Die Frametime wird mit dem Kehrwert des Deltatime ermittelt und schließlich als 2D – Text ausgegeben.



F3 – Wireframe – Modus

wird mit Hilfe von `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)` eingeschaltet und mit `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)` ausgeschaltet

F4 – Cel Shading

wird damit ein und ausgeschaltet

F8 - View Frustrum Culling

wird damit ein und ausgeschaltet und dabei wird mit einer Flag gearbeitet

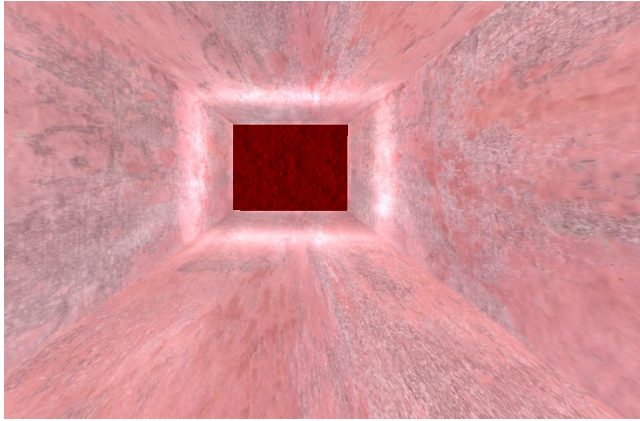
Illumination & Texture

Grundsätzlich wird jedes Objekt, außer das Wasser, die Procedural Textures durch Pointlights beleuchtet.

Im Level 2 gibt es 2 Gänge, die mit Hilfe von Light Maps beleuchtet werden. Der erste Gang ist gleich am Anfang des Levels.



Der zweite Gang ist gleich nach diesem Gang auf der rechten Seite. Hier sieht man dann auch deutlich die Specular - Lichtkomponente.



Jedes sichtbare Objekt hat eine Textur. Meistens sind die Texturen PNGs oder Bitmaps. Wie bereits erwähnt gibt es auch Wände, die mit Hilfe von dem Perlin Noise – Algorithmus generiert wurden. Auch existiert eine 1D – Texture, die zufällige Werte beinhaltet. Diese wird für das Particle – System genutzt, um zufällige Richtungen simulieren.

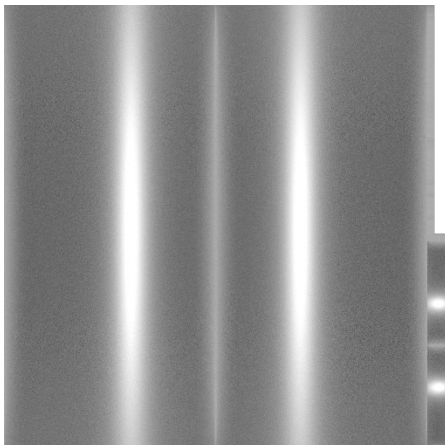
Additional Libraries

- *Assimp* für Mesh Loading
- *Bullet* für die Physik – Simulation und Collision Detection
- *STBI* für Bilder Laden
- *FreeType* für 2D - Schrift

Effect Implementation

Light Mapping & Separate Texture

Gänge im Level werden mit Hilfe von Blender beleuchtet. Dabei werden Pointlights in Blender positioniert und für den selektierten Gang wird eine Light Map baked. Diese bekommen dann zusätzlich einen neuen UV-Kanal. Zum Beispiel:



Beim Export wird als FBX – Datei exportiert, da diese mehrere UV-Kanäle unterstützt. Dieser Kanal wird für die Lightmaps genutzt. Nur die Gänge, die mit Hilfe

von Blender beleuchtet werden, haben extra UV-Kanäle. Die Lightmaps werden dann als separate Textur genutzt und in den Shader für die Berechnung übergeben.

Heads-Up Display

2D-Texte werden als Bitmaps gerendert. Dabei werden Vertex – Buffers genutzt, um Informationen von der Größe der Schrift zu speichern. Beim Rendern von einem Schriftzeichen werden 6 Positionen & Uv - Koordinaten genutzt. Durch 2 Parameter können die Breite und Höhe verändert werden, in dem diese zu den Positionen addiert werden. Mit Hilfe von 2D Koordinaten können dann auch die Position am Bildschirm ausgewählt werden. Dabei hat jedes Character immer eine 2D – Grundposition.

Physics Engine

Wir haben uns für Bullet als Physics Engine entschieden. Objekte in unseren dynamicsWorld sind alle Gegner, das Level, die Vulnerables, die Wände mit prozeduralen Texturen(WPT) und der Spieler selbst.

Wir erstellen aus den Punkten im Level-Mesh *Faces*, die jeweils 3 Vertices beinhalten. Aus diesen Faces erstellen wir uns ein *btBvhTriangleMesShape* welches eine Masse von 0 hat – die Masse ist also unendlich und nicht bewegbar. Die Vulnerables und WPT sind einfacher gestaltet, haben aber ebenso eine Masse von 0.

Die Gegner und der Spieler haben beide eine *btBoxShape*.

Gravity ist unserem Spiel deaktiviert. Die Collision Detection passiert über unserem dispatcher, von dem wir die Manifolds bekommen. Aus den Manifolds holen wir uns die Kontaktpunkte und die zwei Kollisionskörper. Danach können wir vergleichen ob es sich um eine Spieler-Gegner Kollision handelt, oder ob der Spieler an den Levelgrenzen ankommt. Falls eine Spieler-Gegner Kollision passiert, dann verliert der Spieler Lebenspunkte.

GPU-Particle System

Dieser besteht aus im Großen und Ganzen aus 3 Schritten.

Der erste Schritt ist das Instanzieren. Dabei wird eine Richtung, in der ein Partikel fliegen soll und eine Position übergeben. Dann wird eine Anzahl von Partikel mit diesen Werten instanziiert. Danach werden noch Vertex Array - Objects, Vertex - Buffer und Transformfeedback – Buffer erstellt. Als Attribute für werden die Zustände vom Partikel, die Position, die Geschwindigkeit und Lebenszeit übergeben und auch gleich mit den übergebenen Werten, aber auch mit Default - Werten befüllt.

Das Rendern besteht dann nur noch aus 2 Schritten, dem Übergeben der alten Werte und Zustände und dem eigentlichen Rendern. Mit Hilfe von den Transformfeedback – Buffer werden im Shader die vorherigen Werte(Zustände vom Partikel, die Position, die Geschwindigkeit und Lebenszeit) übernommen und neue Positionen ermittelt oder neue Partikel generiert. Dann beginnt der Schritt noch einmal. Dabei endet dieser Zyklus, wenn die maximale Lebenszeit erreicht wurde.

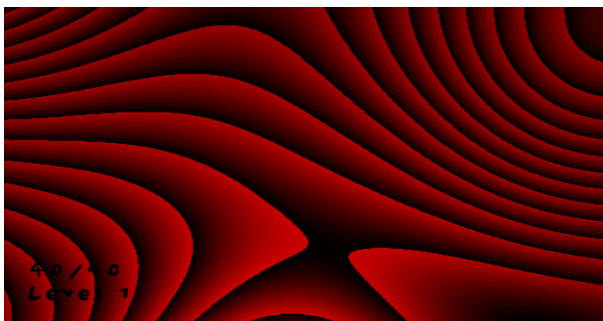
Im letzten Schritt werden die kürzlich errechneten Werte aus dem Transformfeedback – Buffer herangezogen, um die roten Particle zu rendern. Diese haben eine einheitliche Größe.

Water Mesh

Um Wasser zu simulieren, wird alles drei Mal gerendert. Die ersten zwei Renderrunden werden alle in separate Framebuffer gerendert. Dabei werden alle Objekte, wie bereits erwähnt, gerendert, wobei das Water – Mesh nicht gerendert wird. Diese 2 Framebuffers werden dann auf einer Textur gerendert und aus dieser wird dann die Wasserreflektion im Shader berechnet. Dabei ist die y-Achse gespiegelt.

Procedural Textures

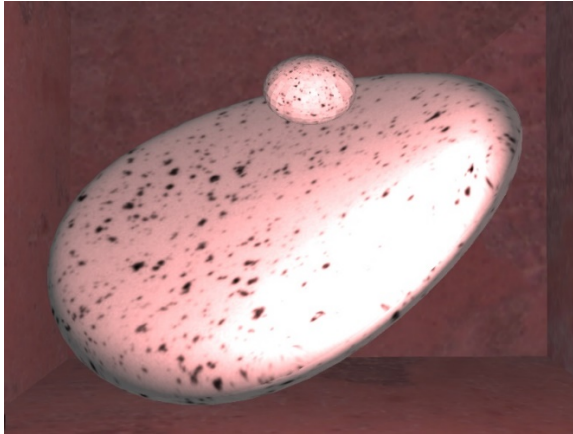
Die Farbe für die Textur wird mit Hilfe vom Perlin Noise – Algorithmus berechnet. Im Grunde werden alle 3D – Positionen auf einem Punkt im Einheitswürfel abgebildet. Für die Einheitsvektoren $((1,1,0),(-1,1,0),(1,-1,0),(-1,-1,0),(1,0,1),(-1,0,1),(1,0,-1),(-1,0,-1),(0,1,1),(0,-1,1),(0,1,-1),(0,-1,-1))$ werden Pseudo – Gradientenvektoren errechnet und der Distanzvektor vom Eingabewert zu den einzelnen Einheitsvektoren ermittelt. Daraufhin wird mit Hilfe von 8 Distanzvektoren der Punkt ermittelt. Anschließend wird das Skalarprodukt vom Gradienten - und Distanzvektor berechnet. Diese 8 Werte werden dann nur noch interpoliert. Die Texture dann folgendermaßen aus:



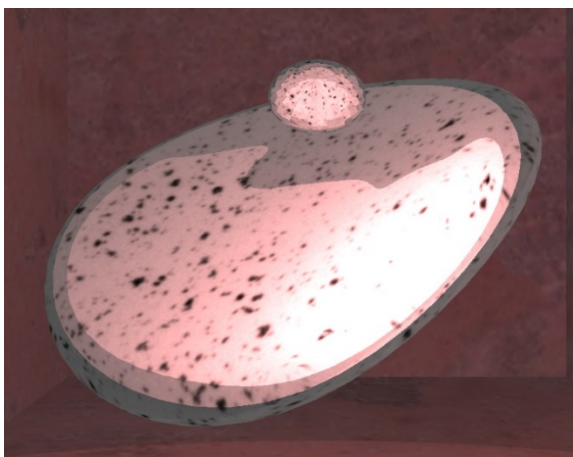
Cel Shading

Kann mit der *F4* – Taste ein und ausgeschalten werden. Diesen Effekt sieht man vor allem am Gegner.

Zum Vergleich ohne Cel – Shading



Mit Cel Shading



Im Shader gibt es eine Variable *levels*. Diese wird einfach mit der Helligkeit multipliziert, gerundet und schließlich durch *levels* dividiert, um diesen Effekt zu erzielen.

Der Wert von Levels kann auch im Config – File eingestellt eingestellt werden.

Effects References

- HUD: <https://learnopengl.com/In-Practice/Text-Rendering>
- Water-Mesh: https://www.youtube.com/watch?v=HusvGeEDU_U&list=PLRIWtICgwaX_23jiqVByUs0bqhnaINTNZh
- GPU-Particle System: <http://ogldev.atspace.co.uk/www/tutorial28/tutorial28.html>
- Procedural Textures: <https://flafla2.github.io/2014/08/09/perlinnoise.html>

Special Features

Es sind keine speziellen Features implementiert.

Complex Interactions

Die einzige Interaktion ist, die existiert, ist, wie bereits erwähnt, der Angriff auf die Wände. Dabei muss man Nahe genug an der Wand sein, um diese anzugreifen. Ein Angriff wird mit der *Space* – Taste gestartet, wobei man die Taste gedrückt halten muss. Ein Angriff ist dann erfolgreich und ersichtlich, wenn in der Mitte der Wand kleine rote Partikel strömen und die Wandfarbe langsam verschwindet.