

# Dokumentation - Abgabe 2

186.831 UE Computergraphik 2019 SS

Florian Holzmann (11776181)

Martin Steinmetz (11777767)

19. Juni 2019

## Beschreibung

Dimensionless ist ein First Person 3D Geschicklichkeits-Spiel, indem man in einer First-Person-Perspektive durch die Map navigieren muss. Um das Ziel zu erreichen muss man geschickt zwischen zwei Dimensionen wechseln und somit nicht aus der Map zu fallen oder sich den Weg frei zu räumen.

Eine Map kann in Map-Files gespeichert werden und so in das Spiel geladen werden. Ein Spieler startet auf einer im Map-File definierten Position und kann mittels Tastatur und Maus bewegt werden. Auf den Spieler wirkt eine dauerhafte Schwerkraft, welche ihn Richtung Boden zieht. Mit der Dimensionstaste kann zwischen den beiden Dimensionen gewechselt werden.

Fällt man von der Karte wird man entweder zurück zum Start oder zum zuletzt passiertten Checkpoint teleportiert. Um zu gewinnen muss man den Zielbereich am Ende der Karte erreichen.

Die Implementierung von Dimensionless baut auf dem uns zur Verfügung gestelltem Framework auf, welches wir mit den benötigten Spiel-Klassen erweitert haben. Außerdem integrierten wir die Physics-Engine Nvidia PhysX in der Version 4.1 [1][2]. Die Physics Engine wird zur Kollisionserkennung, teilweise im Character-Controller und für die Schwerkraft verwendet. Zusätzlich ermöglicht sie auch den Doppel-Sprung und den Wallrun.

## Features

### Effekte

#### Cel Shading

Wir entschlossen uns Cel Shading zu verwenden um unserem Spiel einen Cartoon-Look zu geben. Implementiert wurde das Cel Shading durch eine Stufung der brightness im fragment Shader (texture.frag). Zusätzlich haben wir noch die Farben der Texturen in das HSV-Bildformat umgerechnet und dort ebenfalls die Brightness gestuft. Dies hat zur Folge, dass die Texturen ebenfalls einen cartoon-look aufweisen. Sichtbar ist der Effekt bei allen Texturen und die grundsätzliche Stufung der brightness vor allem im Start-Raum. Für die Video textures haben wir einen zweiten Shader erstellt (textureVideo.frag), weil die Video Texturen durch die Veränderungen an der Texturfarbe unbrauchbar werden.

#### CPU Particle System

Um den Wechsel zwischen den beiden Dimensionen etwas interessanter zu gestalten haben wir ein CPU Particle System implementiert. Jedes mal wenn der Spieler die Dimension wechselt öffnet sich vor ihm eine art Portal. Hierbei werden eine bestimmte Anzahl and individuellen Partikeln erstellt. Jedes einzelne Partikel besitzt eine durch den Zufall beeinflusste TTL(Time-To-Live), Richtung und Geschwindigkeit. Partikel, die zusammengehören, werden alle in einem System zusammengefasst und bestehen nur aus einer grafischen Darstellung und keinen eigenen PhysX Objekten, da dies zu einem zu starken Leistungseinbruch des Spiels führen würde. Zusätzliche Verwendung findet das Particle System im End-Raum, in den man kommt, wenn man das Spiel gewonnen hat. Hier wird es als Feuerwerk verwendet. Die Implementierung bedindet sich in den Klassen Particle.cpp und ParticleSystem.cpp.

#### Video textures

Die Video Textures finden 2 Anwendungsbereiche. Einerseits im Tutorial-Video im Anfangsraum und als Lava in den Abgründen der Map. Prinzipiell wurden beide Video Textures auf die selbe Weise umgesetzt. Wie haben uns im Allgemeinen dazu entschieden nicht die Video Dateien direkt sondern nur dessen Frames in das Programm zu laden. Zunächst wurde hierfür in Adobe After Effects das Video erstellt und als AVI(Raw) Datei gespeichert. Anschließend wurde das Video mithilfe von Blender in seinen einzelnen Frames als PNG Dateien gespeichert. Diese wurden dann noch mithilfe eines Konverters in DDS-Files umgewandelt. Diese Dateien werden im Programm nacheinander auf ein und die selbe Textur geladen um ein Video darzustellen. Wichtig ist auch, dass immer nur so viele Frames in einer Sekunde geladen werden, wie auch das Video an framerate aufweist. Dies wurde mithilfe einer einfachen Berechnung über die Delta-Time umgesetzt. Die Implementierung befindet sich in der Klasse VideoTexture.cpp.

## **Hierarchical Animation**

Um die Hindernis Vielfalt eines Levels zu erweitern wurden Hierarchical Animations implementiert. Der Effect wird sowohl bei Hindernissen als auch bei Plattformen angewandt. Dabei besitzt jedes Objekt eine Animation. Weiters kann ein Objekt als Parent eins oder mehrerer anderen definiert werden. Durch dieses Binden eines Objektes an ein Parent-Objekt summiert sich die Animation des jeweiligen Kind-Objektes auf und eine Hierarchical Animation entsteht.

Die Daten für diesen Effect wird aus einem Map-File (.mdyn) gelesen. Dieses besteht aus Geometriedaten der Objekte als auch aus Listen von den Punkten der Animationen, sowie Definitionen der Parent-Objekte. Das File wird in der Map Klasse geladen und die jeweiligen Geometrien inklusive Collider erstellt. Die jeweiligen Punkte der Animationen werden in Dauerschleife von ihren Objekten abgefahren.

Ein Objekt mit Hierarchical Animation wird durch die Klasse Drawable.cpp repräsentiert. Zu sehen ist der Effect bei der bewegten Plattform nach dem Startbereich und bei dem Laser-Paar im Gang des hinteren Bereiches der Map.

## **Contours via backfaces**

Um den Cartoon-Look unseres Spieles zusätzlich zum Cel-Shading noch zu verstärken, entschlossen wir uns Contours via backfaces zu implementieren. Dabei werden die normalerweise ausgeblendeten Back-Faces aller Geometrien extra gezeichnet. Sie werden dabei um einen dynamischen Faktor skaliert und schwarz gezeichnet. Dies hat eine schwarze Umrandung aller Objekte zur Folge.

Der Effect wurde in der Klasse Geometry.cpp in der draw Methode implementiert. Am besten zu sehen ist der Effect bei dem bewegten Hindernis auf der Plattform oder bei den Plattformen im großen Raum in der Mitte der Map.

## **Andere Features**

### **PhysX**

Als Physics-Engine verwenden wir Nvidia PhysX 4.1 [1][2].

Der Engine wurde ins Framework integriert und dient nun zur Kollisionsbehandlung zwischen Player und Map, sowie dynamischen Objekten. Außerdem definiert PhysX eine Schwerkraft, welche auf den Spieler und dynamische Objekte wirkt. Aktuell wird nur das Springen des Players mittels PhysX-Kräften realisiert, nicht jedoch die Bewegungen.

## **Map Loader**

Karten werden in eigens konstruierten Map-Files abgespeichert und können Mittels Map Loader in das Spiel geladen werden.

Eine Karte besteht aus fünf Dateien (.map, .mdim1, .mdim2, .mdyn, .mvid). Das MAP File definiert Startposition, Zielposition, Check-Points, Lowest-Point, sowie Positionen

und Parameter der Spotlights. MDIM1/2 basieren auf OBJ-Files und beschreiben die Geometrien und Texturen der beiden Dimensionen. MDYN basiert ebenfalls auf einem OBJ-File und definiert dynamische Objekte, welche in beiden Dimensionen zu finden sind. zusätzlich zu den Geometriedaten befinden sich auch Listen der Punkte für die jeweiligen Animationen, sowie Definition den jeweiligen Parent-Objekt in der Datei. MVID definiert die Objekte auf denen Video-Texturen angewendet werden. Sie beinhalten wiederum Geometriedaten, sowie den Name und die Anzahl der Frames.

Der Map-Loader lädt diese Dateien nach und nach ein, und überprüft jeweils ob die Dateien fehlerhaft waren. Ist dies der Fall wird eine Fehlermeldung ausgegeben und das Programm beendet.

Map-Dimensionen bestehen aus den Render-Objekten, um die Map zu zeichnen, und aus PhysX-Objekten, was die Collider der Map sind. Um den Dimensionswechsel zu realisieren besitzt jede der beiden Dimensionen eine eigene PhysX-Szene.

Der Maploader wurde in die Klasse Map.cpp integriert.

## **Dimensionswechsel**

Nach vollständigem laden einer Map kann mit der F-Taste zwischen den beiden Dimensionen gewechselt werden. Beide Map-Dimensionen existieren parallel, werden jedoch nie gleichzeitig gezeichnet. Zusätzlich wird der Spieler der jeweiligen PhysX-Szene zugewiesen und von der anderen entfernt. Somit werden jeweils nur Kollisionen mit der aktuellen Dimension beachtet.

Bei einem Dimensionswechsel werden als visueller Effekt Partikel mittels Particle-System generiert und angezeigt.

Der Dimensionswechsel wurde in der Klasse Map.cpp in der switchDimension Methode implementiert.

## **Character Controller**

Um den Spieler durch die Karte zu navigieren, haben wir einen Character-Controller in die Player-Klasse implementiert. Der Spieler kann sich mit den Tasten W, S, A und D nach Vor, Zurück, Links und Rechts bewegen und mit der Leer-Taste Springen. Durch nochmaliges Betätigen der Leertaste in der Luft kann ein Doppel-Sprung ausgeführt werden. Die Blickrichtung des Spielers wird mit der Maus gesteuert. Bewegung erfolgt durch Positionsänderung, das Springen hingegen wird durch die Anwendung einer Kraft, des Physics-Engin, realisiert. Mittels Polling wird in jedem Frame auf Tasten überprüft und eine Move-Methode des Players aufgerufen, welche die Position des Spielers aktualisiert. Dabei wird sowohl die Position der Kamera, als auch des Colliders, des Spielers, verändert. Zusätzlich ist es möglich, dass der Spieler an der Wand laufen kann. Sobald der Spieler eine Wand berührt ist es ihm möglich 5 Sekunden an der Wand zu laufen und auch von dieser abspringen. Danach bekommt er einen Cooldown, das bedeutet, dass er für 3 Sekunden nicht mehr an der Wand laufen kann. Die Wand- und Bodenerkennung wurde mit Raycasts realisiert.

Controls	
Key:	Action:
W	Vorwärts
S	Rückwärts
A	Links
D	Rechts
L-Shift	Sprinten
Space	Springen
F	Dimensionswechsel
Maus	Kamera bewegen
Dev-Controls	
F1	Wireframe-Mode
X	No-Clip Mode

## Win/Lose

Ein Spieler kann nur dann verlieren, wenn er durch eines der Löcher in der Map fällt und somit eine bestimmte (negative) Höhe erreicht. Tritt dieser Fall auf, wird der Spieler nach Ablauf von zwei Sekunden zurück zum Startpunkt bzw. des zuletzt erreichten Checkpoints teleportiert.

Gewonnen hat ein Spieler genau dann, wenn dieser das Ziel, in Form eines türkisen Affenkopfes, am Ende der Map erreicht hat. Wurde dies geschafft, so wird der Spieler in einen Endraum teleportiert. Dort kann er ein Feuerwerk bestaunen und wird nach 10 Sekunden wieder an den Start des Spiels befördert, wo er sich nochmals an diesem versuchen kann.

## Belichtung und Texturing

An der Belichtung wurde am Framework nicht viel verändert. Die gesamte Szene wird durch ein Directional-Light und einigen Point-Lights belichtet. Die Anzahl und Position der Point-Lights kann durch das MAP-File bestimmt werden.

Es gibt prinzipiell immer 2 Map Dateien. Die eine davon weist Texturen auf, die einen alten ägyptischen Tempel darstellen sollen. Die andere einen dunklen und bedrohlichen Wald. Texturen werden immer (außer bei den Video Textures) in den Map-Files gesetzt.

## Externe Quellen und Tools

Es wurde lediglich der Physics-Engine NVIDIA PhysX 4.1 [1][2] als externe Quelle verwendet.

Die Map wurden in Blender [3] modelliert und mit einem Texteditor bearbeitet.

## Literatur

- [1] PhysX GitHub Repository. *<https://github.com/NVIDIAGameWorks/PhysX>* 2019.
- [2] PhysX Doku. *<http://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html>* 2019.
- [3] Blender. *<https://www.blender.org>* 2019.