

# Sun'n Slaughter Documentation

Sebastian Karall / 01635881  
Aleksandar Marinkovic / 01634028

June 2021

# 1. Implementation

## 1.1. Terrain Tessellation

Das Terrain wird als 2D-Plane während der Laufzeit erstellt. Im Vertex Shader werden die Höhendaten aus einer Heightmap ausgelesen und dem Terrain als neue Höhe zugewiesen. Der Grad der Tessellation basiert auf der Entfernung von der Kamera zur Oberfläche des Terrains, die Berechnung erfolgt im Tessellation Control Shader. Im Tessellation Evaluation Shader werden die neuen Positionen sowie die Normalen für das Terrain bestimmt.

## 1.2. Shadow Map with PCF

Um eine Shadow Map zu generieren, muss zuerst ein FBO erzeugt werden, welche die Depth der Szene speichert. Die Shadow Map wird aus der Sicht der Lichtquelle gerendert, unsere Lichtquelle ist die Sonne deswegen kann angenommen werden, dass alle Lichtstrahlen parallel zueinander sind. Die Szene muss zweimal gerendert werden, beim ersten Durchgang wird die Depth aller Objekte in der Szene aus der Sicht der Lichtquelle in die Shadow Map gespeichert. Beim zweiten Durchgang wird alles ganz normal gerendert und die Shadow Map als eine uniform sampler2D an den Fragment Shader übergeben. Der Shader ist für die Berechnung der Schatten sowie für das Korrigieren von Artefakten, wie Peter Panning und Shadow Acne zuständig. Damit die Shadow Map nicht so "blockig" aussieht, werden die benachbarten Texels beim Berechnen der Schatten miteinbezogen (PCF).

## 1.3. Cel Shading

Im Terrain und Texture Fragment Shader werden die RGB-Farbwerte zu HSV-Werten transformiert, danach wird der Value-Wert mittels einer Variable in n verschiedenen Level diskretisiert.

## 1.4. Lens Flares

Lens Flares werden mittels des GUI-Renderers auf den Bildschirm in Screen-Space gerendert. Die Position und Intensität der Flares ist durch die Distanz zwischen der Lichtquelle (Screen-Space) und dem Mittelpunkt des Spielfensters definiert. Außerdem wird eine Query verwendet, die überprüft, ob und wie viel der Lichtquelle durch ein anderes 3D-Objekt bedeckt ist. Je nach Deckungsgrad wird die Intensität und Transparenz der Lens Flares angepasst.

## 1.5. Skybox

Auf der Innenseite des Würfels werden Texturen des Horizonts abgebildet, um das Level größer erscheinen zu lassen.

## 1.6. Keyframe Animation (Hierarchical Animation)

Die Spielfigur besitzt mehrere Posen, welche in Blender erstellt wurden. Ein Compute Shader interpoliert zwischen den Posen, um eine Laufanimation darzustellen.

## 1.7. Frei bewegliche Kamera

Diese wurde als observierende 3rd-Person-Kamera mittels Maussteuerung und den WASD-Tasten zur Steuerung der Spielfigur umgesetzt. Sie verfolgt sozusagen den Hauptcharakter.

## 1.8. Bewegliche Objekte

Es gibt mehrere bewegliche Objekte, allen voran die Spielfigur welche mittels WASD-Tasten und der Maus gesteuert werden kann, sowie die zahlreichen Gegner, welche die Spielfigur verfolgen. Dies geschieht unter Einsatz der Kollisionsdetektion von Nvidia PhysX.

## 1.9. Texture Mapping

Die UV Koordinaten sind von den importierten Modellen vorgegeben. Angepasste Meshes wurden in Blender neu auf die Texturen gemappt und ebenso durch assimp in die eigene Datenstruktur geladen.

## 1.10. Mesh Cooking

Aus Vertices und deren Index werden Formen mit PhysX modelliert, welche zur Kollisionsdetektion verwendet werden können. Somit sind komplexe Boden- und Wandformen im Spiel möglich.

## 1.11. Position Disk Sampling

Position Disk Sampling wird verwendet, um zum Beginn des Spiels die Positionen der Bäume zu erzeugen. Die Bäume werden zufällig erzeugt und besitzen einen vordefinierten Mindestabstand zueinander. Außerdem überlappen sie sich nicht.

## 1.12. GPU Partikel mit Compute Shader

Die Dash-Attack "wirbelt Staub auf" welcher mittels eines Compute Shader berechnet wird. Dazu werden die Positionen der Partikel berechnet und im Anschluss einem Vertex/Geometry/Fragment Shader übergeben, um eine Textur darauf zu zeichnen.

## 1.13. View Frustum Culling

Um bessere Performance zu gewinnen, werden Objekte welche sich nicht im View Frustum befinden erst gar nicht an die GPU übergeben.

## 1.14. Weitere Implementationen wie z.B.

... FPS Limiter, Sound, WorldBoundaries, CollisionDetection, Gui/TextRenderer etc.

## 2. Features

### 2.1. Terrain

Terrain basierend auf einer Heightmap, Tessellation Grad in Abhängigkeit auf der Distanz zwischen Kamera und Oberfläche des Terrains.

### 2.2. Shadows

Dynamische Schatten basierend auf der Position der Lichtquelle.

### 2.3. Lens Flares

Beim Betrachten der Lichtquelle werden Lens Flare Texturen auf dem Bildschirm gerendert.

### 2.4. Basic Gameplay / Advanced Gameplay

Die Spielfigur kann sich frei bewegen und es können den Gegnern Lebenspunkte durch verschiedene Aktionen abgezogen werden. Die Gegner selbst werden mit der Zeit immer stärker und verfolgen die Spielfigur.

### 2.5. HUD

Die Lebenspunkte der Spielfigur sowie andere Werte werden in 2D als HUD über dem eigentlichen Spielgeschehen angezeigt.

### 2.6. Partikel

Beim Ausführen der Dash-Attacke lässt die Spielfigur eine GPU-Partikel-Staubwolke hinter sich.

### 2.7. View Frustum Culling

Nur Objekte die sich im View des Spielers befinden werden gerendert

### 2.8. Hierarchical Animation

Die Spielfigur wird beim Gehen animiert.

### 2.9. Cel Shading

Im Terrain & Texture Fragment Shader werden Farbwerte diskretisiert.

### 2.10. Hierarchical Animation

Die Spielfigur wird beim Gehen animiert.

## 3. Libraries

### 3.1. Image-Loader

Der header stbimage.h wurde verwendet, um diverse Texturen zu laden.

URL: [https://github.com/nothings/stb/blob/master/stb\\_image.h](https://github.com/nothings/stb/blob/master/stb_image.h)

### 3.2. Nvidia PhysX 3.4

Mit PhysX wird die allgemeine Kollisionsdetektion, sowie die Bewegung von Spielfigur und NPCs innerhalb der Spielwelt umgesetzt.

URL: <https://github.com/NVIDIAGameWorks/PhysX-3.4>

### 3.3. Assimp

Mit assimp kann eine Vielzahl von 3D-Objekten importiert werden.

URL: <http://www.assimp.org/>

Das Konzept des Imports wurde von folgendem Tutorial übernommen:

<https://learnopengl.com/Model-Loading/Assimp>

### 3.4. FreeImage

Ermöglicht das Lesen von gängigen Grafikbildformaten wie PNG, BMP, JPEG, TIFF und andere. Da die Texturen nicht manuell umgewandelt werden müssen, ist es einfacher Modelle mit existierenden Texturen durch "assimp" zu importieren.

URL: <http://freeimage.sourceforge.net/>

### 3.5. FreeType

Innerhalb des Spiels wird FreeType zum Anzeigen des HUD und etwaiger Text-Rückmeldungen in 2D genutzt.

URL: <https://www.freetype.org/>

### 3.6. irrKlang

Innerhalb des Spiels wird irrKlang zum Abspielen von Sounds und etwaiger Musik genutzt.

URL: <https://www.ambiera.com/irrklang/>