

# Erelas - Dokumentation

Amir El Agrod (01631844)  
Eva-Maria Resch (01633055)

Bei 'Erelas' handelt es sich um ein Geschicklichkeitsspiel. Die Spielfigur kann kontrolliert und Bretter können abgelegt werden.

Deine Stadt wurde durch einen Überraschungsangriff von Orks zerstört. Sie wollen alle südlichen Städte unter ihre Kontrolle bringen. Du bist der einzige Überlebende/die einzige Überlebende und möchtest die Nachbarstädte warnen. Da das Heer die Hauptstraßen kontrolliert, fliehst du über einen verwahrlosten Bergweg, um auf dem Erelas ein Warnfeuer zu zünden. Aber Achtung! Der Bergpfad ist zerklüftet. Nur über verfallene, sich bewegende Brücken kannst du die Flusstäler überwinden. Da dein Bein angeschossen wurde, kannst du nicht springen und musst durch Niederlegen eines Brettes einen Weg über die zerstörten Brücken finden.

## 1 Features

- Physics-Engine - Kollisionsobjekte für alle Spielobjekte erstellt
- GPU-Particle-System
- HDR rendering
- prozedurale Textur
- animiertes Wassermesh
- Heads-up-Display - wichtige Spielinformationen werden angezeigt
- View-Frustum-Culling
- Skybox
- Brettmodus
- Timer
- 3 Level, Schalten zum nächsten und Wiederholen des aktuellen Levels möglich
- Godmodus - Gravity Dis/enabling mit Tastendruck



Abbildung 1: Level1



Abbildung 2: Level2



Abbildung 3: Level3

## 2 Technische Details

### 2.1 Unterschiedliche Objekte im Spiel

- Berglandschaft: aus Heightmap generiertes Terrain, in Blender verfeinert und aus obj.-File geladen
- Brücken: animierte Meshes - als Cubes gezeichnet, eine Brücke je Level als komplexeres geladenes Objekt aus obj.-File
- Holzbrett: grafisches Primitiv - Cube
- Fackel: Feuer mit GPU-Particle-System dargestellt, Texturatlas verwendet, um realistischere Optik zu erzielen
- Warnfeuerstelle: Cube mit darauf platzierter aus einem obj.-File geladener Feuerschüssel, mit prozeduraler Textur versehen
- Wasser: Wassermesh, animiert mit Vertex-Displacement

### 2.2 Collision Detection

Objektkollisionen werden mithilfe der Physics Engine PhysX von NVIDIA berechnet. Dafür wird ein Bounding Volume verwendet, das anhand von bestimmten Objekteigenschaften eine Kollision der physikalischen Formen der Objekte erkennt.

Für alle statischen Objekte werden PhysX-Geometrie-Objekte des Typs 'PxRigidStatic' verwendet. Auch für die Brücken wurden statische Kollisionsobjekte angelegt, da sie ihre Position bzw. Bewegungsbahn nie ändern sollen. Mit Hilfe des von der Klasse PxUserControllerHitReport abgeleiteten Hit-Reports "CustomControllerHitReport" wird auf Kontakte des Character Controllers mit sich in X- und Z-Richtung bewegendes Brücken überprüft. Besteht Kontakt wird der Character Controller dementsprechend mit der Brücke mitbewegt.

Das Kollisionsobjekt des Terrains ist ein aus den Daten der Terraingemetry erstelltes Heightfield.

Um die Bretter korrekt abzulegen, wurden diese als 'PxRigidDynamic' angelegt und die Position der tatsächlich gezeichneten Brett-Objekte an die Position der zugehörigen PxGeometry-Objekte angepasst. So werden Verschiebungen durch Kollisionen berücksichtigt und das Fallen der Bretter sieht realistisch aus. Um sich die Erweiterung des Frameworks um Sound offen zu halten, wurde ein von der Klasse PxSimulationEventCallback abgeleitetes 'CustomSimulationEventCallback' angelegt, das überprüft, wann dynamische PxBoxGeometry-Objekte mit anderen Physics-Kollisionsobjekten kollidieren. Im Fall unseres Spieles wären das Berührungen der Bretter mit anderen Spielobjekten. Hier könnte man später das Abspielen eines Soundgeräusches für das Fallen der Bretter einbauen und in genau jenen Berührungssituationen abspielen.

Die Spielfigur bekam als Kollisionsobject einen Character Controller, dessen Kollisionsreaktionen mit dem CustomControllerHitReport spezifiziert werden.

Für die Wasseroberfläche des Flusses wurde eine Plane angelegt. Damit der Spieler/die Spielerin nicht über den Terrainrand hinausgehen kann, wurden um es herum Planes als Spielfeldbegrenzung platziert.

Für die ganze Szene gilt, wenn aktiviert, eine Schwerkraft, sodass die Kameraposition sich an der Y-Achse verringert - der Spieler/die Spielerin fällt, wenn die Spielfigur sich nicht auf dem Terrain oder einem Objekt befindet.

## **2.3 Komplexe 3D-Modelle**

Zum Laden komplexer Modelle aus obj.-Files haben wir die Model-Loading-Library Open-Asset-Importer-Lib in das Framework eingebaut. Die von uns verwendeten Modelle wurden von 'www.turbosquid.com' heruntergeladen und sind lizenzfrei verwendbar. Es wurden je Level das Terrain, eine Brücke und eine Feuerschüssel geladen. Keine der von uns verwendeten komplexen Objekte sind selbst erstellt.

## **2.4 Effekte**

### **2.4.1 GPU-Particle System (+ Compute Shader, Instancing)**

Um das Feuer unserer Fackel im Spiel darzustellen, wurde ein GPU-Particle System implementiert. Dem VertexShader wird die Position des Partikel-Emitters übergeben, im Geometry Shader die UV-Koordinaten für die Textur ermittelt - es wurde ein Texturatlas verwendet, um die Unregelmäßigkeiten der Form eines Feuers nachzuempfinden. Die Positionen der einzelnen Partikel werden mit einem Compute-Shader berechnet. Bei der Umsetzung von Instancing gab es Probleme. Im Spiel wurden die Partikel ohne Instancing erstellt. Die Partikel werden ebenfalls für die Darstellung des Feuers der Feuerstation im Ziel verwendet.

Quellen: <http://www.geeks3d.com/20140815/particle-billboarding-with-the-geometry-shader-gsl/>, Folien des Repetitoriums zu Particle Systems

### **2.4.2 Water Mesh**

Der Fluss in unseren Levels wurde mit einem Wassermesh dargestellt. Mittels Displacement-Mapping wird die Vertex-Position durch eine Sinusfunktion verschoben. Zusätzlich wurden durch Rendern der Szene in ein Frambufferobjekt eine Reflection- und Refraction-textur erstellt und mit projective Texturing auf die Wasseroberfläche aufgetragen. Die Reflection-Texture bewegt sich aus uns unbekannten Gründen mit der Cursorposition mit und scheint somit optisch zu verrutschen.

Quellen: Youtube-Tutorial-Serie [https://www.youtube.com/watch?v=HusvGeEDU\\_U&list=PLRIWtICgwaX23jiqVByUs0bqhna1NTNZh](https://www.youtube.com/watch?v=HusvGeEDU_U&list=PLRIWtICgwaX23jiqVByUs0bqhna1NTNZh), Folien des Repetitoriums zu Wassermeshes

### 2.4.3 Procedural textures

Für unsere Feuerstation wurde mit prozeduraler Texturierung eine Granitstruktur nachempfunden. Es wird eine 3D-Texture-Map erstellt, in der eine Simplex-Noise-Funktion gespeichert wird. Im Fragment Shader wird daraus die Textur errechnet.

Quellen: OpenGL Programming Guide - The Official Guide to Learning OpenGL, Version 4.5,

Simplex Noise übernommen von <https://github.com/DynamoDS/designscript-archive/blob/master/Libraries/Experimental/SimplexNoise.cs>

### 2.4.4 HDR rendering und Tone mapping

Für das High Dynamic Range Rendering wird die Szene in einen Float-Framebuffer gerendert und als Textur an die Shader übergeben. Im Fragment-Shader wird mit Hilfe der Berechnung der average Luminance ein Exposure-Wert für jedes Texel errechnet und darauf angewandt. Die erhaltenen Werte werden auf Low Dynamic Range zurückgerechnet und die Textur als screensized Quad in das Spielfenster gerenderd. Um Aialiasing zu vermindern wurde ein Multisample-Buffer erstellt.

Quellen: <https://learnopengl.com/Advanced-Lighting/HDR>

Antialiasing: <https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing>

Shader orientiert an: OpenGL SuperBible, Seventh Edition



Abbildung 4: Szene mit HDR Rendering



Abbildung 5: Szene ohne HDR Rendering

#### 2.4.5 Heads-Up-Display unter der Verwendung Blending

Unter der Verwendung der FreeType-library wurden mit Hilfe von Blending Textausgaben im Spielfenster erzeugt. Hierfür wurden 2 Fonts verwendet: Einerseits ein Herr der Ringe-Font für die Textausgaben zur Anzahl der verfügbaren Bretter, des aktuellen Levels und Ausgaben zur Win-Lose-Situation sowie anderen Spielangaben, andererseits wurde ein Symbolfont verwendet, um mit kleinen Feuersymbolen die verbleibende Spielzeit darzustellen.

Quellen: <https://learnopengl.com/In-Practice/Text-Rendering>

#### 2.4.6 Physics-Engine

Eine genauere Beschreibung befindet sich im Absatz 'Collision Detection'.

Quellen: <https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Index.html>, Code Snippets und Samples von NVIDIA, Learning Physics Modeling with PhysX verfasst von Krishna Kumar

#### 2.4.7 Lightmapping

Lightmapping einzubauen war zeitlich und aufgrund eines Problems mit den UV-Koordinaten nicht mehr möglich. Das Vorgehen war wie folgt: Zuerst wurde jedes Level mit allen statischen Objekten in Blender nachgestellt. Allen in der Szene enthaltenen Objekten - abgesehen von der Feuerstation, da diese mit einer prozeduralen Textur versehen ist - wurden Texturen und Materialien zugewiesen. Um das Directionallight zu simulieren, wurde eine Sonne angelegt. Die Kamera wurde so eingestellt, dass die ganze Szene sichtbar gerendert wird. Die sich daraus ergebende Beleuchtungsinformation wird mit Hilfe des Lightmappacks in die Textur gebaked. Bei Auftragen dieser gab es Probleme, weshalb Lightmapping nicht in das Spiel übernommen wurde. Die nachgestellten Levels in Blender befinden sich im Git-Repository.

Quellen: Folien des Repetitoriums zu Light-Mapping

### 3 Implementierung

#### 3.1 Kamera

Das Spiel wird in der Ego-Perspektive gespielt. Die Welt wird durch die Augen der Spielfigur gesehen. Dabei orientiert sich die Kamerasicht an der Position des Mauszeigers und der Steuerung durch bestimmte Tasten.

Die Kamera wurde als eine First-Person-Kamera umgesetzt und ist mit den unter Steuerung vermerkten Tasten vorwärts, rückwärts, nach rechts und nach links frei bewegbar. Das Drücken von 'G' ermöglicht ein Toggeln der Schwerkraft. Ist diese deaktiviert, kann die Kamera auch nach oben und nach unten bewegt werden. Zusätzlich kann sie durch die Tasten 'E' und 'R' nach rechts oder links rotiert werden. Mit zusätzlichem Drücken der Shift-Taste, können diese Bewegungen im Lauftempo ausgeführt werden. Die Orientierung der Kamera wird ebenfalls an die Position des Mauszeigers angepasst, wobei die

Winkel so beschränkt sind, dass sich der Spieler/die Spielerin nicht überdrehen kann. Ist die Schwerkraft aktiv, kann man auch durch die Bewegung des Mauszeigers nicht weiter von der Oberfläche abheben, als die Höhe, auf der der Spieler/die Spielerin steht.

Für die Kamera wurde als Kollisionsobjekt ein Character Controller angelegt. Dieser wird vor der Kamera entsprechend den Tasten für Kamerabewegung verschoben und die Kamera dann an dessen Position angepasst - so kann Schwerkraft wirken und auf Positionsveränderungen durch Kollisionen eingegangen werden.

### **3.2 Animierte Objekte und Hierarchische Animationen**

Abgesehen von der Kamera, welche vom Spieler/der Spielerin gesteuert wird, gibt es nur eine Art animierter Objekte: Brücken. Diese können sich entweder auf und ab, oder hin und her bewegen, oder aber auch statisch sein. Für die Animation wird die Brückenposition jeweils um  $\text{'DeltaT' * Tempo}$  verschoben. Tempo ist eine von uns festgelegte Konstante. Der Bewegungsmethode werden unter anderem in Weltkoordinaten die obere und untere Grenze - also höchste und niedrigste bzw. linkeste und rechteste Position - der Brücke übergeben. Die Brückenposition pendelt zwischen diesen beiden Grenzen.

Hierarchische Animationen wurden bei zusammengehörigen, durch eine Lücke getrennten Brückenteilen verwendet. Je eine Brücke pro Level ist mit einem sich bewegenden Hindernis versehen, welches durch die hierarchische Animation sowohl seine eigene Verschiebung nach links und rechts, als auch die Auf- und Ab-Bewegung der Parentbrücke ausführt. Die Physics-Kollisionsobjekte werden entsprechend den hierarchischen Animationen mitbewegt, so kann der Spieler/die Spielerin beispielsweise durch das Hindernis von der Brücke geschoben werden.

### **3.3 Texture Mapping**

Mithilfe der eingebundenen FreeImage-Library können Bilder aller gängigen Dateiformate geladen werden. Aus diesen werden Texturen erstellt. Über `"glTexParameter(...)"` wird Mipmapping und Bilinear-Filtering für die Texturen festgelegt. Für alle Objekte im Spiel werden UV-Koordinaten berechnet, nach denen die Texturen aufgetragen werden. Dies passiert in der Klasse 'Geometry'.

### **3.4 Beleuchtung und Material**

Für jede Textur wird ein Material erstellt, bei dem jeweils die Ambient-, Diffuse-, Specular- und Alpha-Werte festgelegt werden können. Jedes Objekt in unserem Spiel ist mit einem Material versehen.

Im Spiel gibt es vier statische und eine dynamische Lichtquelle. Bei den statischen Lichtquellen handelt es sich zum Einen um ein Directional Light, welches das Licht des Mondes darstellt und als globale Lichtquelle die gesamte Szene beleuchtet, zum Anderen um drei im Terrain positionierte Pointlights. Als dynamische Lichtquelle wird ein Pointlight verwendet, welches in konstantem Abstand zur Kamera bewegt wird und die Leuchtkraft des Feuers der Fackel simuliert. Das Pointlight ist eine lokale Lichtquelle und beleuchtet

jeweils nur die unmittelbare Umgebung seiner Position. Die Verwendung von Arrays im Shader ermöglicht, wenn erwünscht, das Anlegen mehrerer Lichtquellen.

Zu jedem Objekt werden Normalen berechnet. Für das Terrain wurden die Normalen durch ein Mittel mit Normalen der angrenzenden Polygone errechnet.

## **3.5 Steuerung**

### **3.5.1 Allgemeines**

Die Spielsteuerung wurde für Per-frame-Operationen mit Polling implementiert, sodass in jedem Durchlauf der Renderloop die einer Taste zugewiesene Operation durchgeführt wird. Das verwenden wir für die Bewegung der Kamera.

Im Key-Callback sind einmalige Operationen implementiert, wie das Toggeln bestimmter Modi - z.B.: Wireframe, Fullscreen...

Im Mouse-Button-Callback wird festgelegt, wie sich der Brettmodus je nach Tastendruck verhält: Aktivierung, Bedienung - die Brettorientierung verändern, Brett-Ablegen oder Beenden.

Das Frame-Buffer-Size-Callback passt nach Veränderungen des Framebufferobjekts die Viewport-Größe an.

Das Window-Size-Callback wird beim Schalten zwischen Windowed-und Fullscreen-Modus verwendet.

Die weiteren Callbacks beziehen sich auf die Kamerasteuerung.

### **3.5.2 Brettmodus**

Mit einem Klick der rechten Maustaste wird der Brettmodus aktiviert, ein Brett herausgeholt - es erscheint auf dem Display und ein Fadenkreuz angezeigt, welches symbolisiert, dass Bretter abgelegt werden können. Wird die rechte Maustaste erneut gedrückt, wechselt das Brett seine Orientierung in eine der vorgegebenen Möglichkeiten. Sind alle Orientierungen einmal dargestellt worden und die rechte Maustaste wird erneut gedrückt, wird der Brettmodus beendet. Befindet man sich im Brettmodus und drückt die linke Maustaste, wird das Brett in entsprechender Orientierung in einem im Spiel vordefinierten Abstand vor der Kamera im Spiel abgelegt. Nach Ablegen eines Brettes wird der Brettmodus automatisch beendet.

Der Brettmodus wurde so umgesetzt, dass in der Klasse 'Plank' je nach Level festgelegt wird, wie viele Bretter es geben soll - die Brettanzahl ist beschränkt, und diese anschließend erstellt werden. Alle verfügbaren Bretter werden in einem `std::vector` gespeichert. Sobald ein Brett abgelegt wird, wird es zu einem `std::vector` hinzugefügt, in dem sich die zu zeichnenden Bretter befinden und an die korrekte Position im Terrain translatiert. Nach dem Erstellen befinden sich die Brettposition defaultmäßig bei (0, 0, 0).



Abbildung 6: Brettmodus

Im Folgenden befindet sich eine Auflistung aller Spielsteuerungsoptionen und Tastenbelegungen:

<b>Taste</b>	<b>Effekt</b>
W, A, S, D	vorwärts, links, rückwärts, rechts gehen
E, R, Q, F	nach rechts drehen, nach links drehen, hoch, hinunter
G	toggle Schwerkraft
L	in nächsthöheres Level schalten - wenn vorhanden
O	aktuelles Level erneut versuchen - neu starten im aktuellen Level
Shift	laufen
rechte Maustaste	Brett herausholen, Brett drehen, Brettmodus beenden
linke Maustaste	Brett ablegen
F1	toggle Help - Textausgabe im Spielfenster
F2	toggle FPS - Anzeige
F3	toggle Wireframe - Modus
F4	toggle Fullscreen - Modus
F5	toggle Back-Face-Culling
F6	toggle Heads-Up-Display
F7	toggle High Dynamic Range-Rendering
F8	toggle View-Frustum-Culling
esc	Spiel beenden

## 4 Gameplay

Die Kamera (= Spielfigur) wird mit den entsprechenden Tasten über das Terrain bewegt. Um das Ziel zu erreichen muss ein Weg über das Terrain gefunden werden, ohne in die Tiefe zu stürzen. Dafür muss die Spielfigur zum Überwinden von Schluchten über die im Terrain platzierten Brücken gehen. Manche dieser Brücken haben Lücken. Durch

gezieltes Ablegen eines Brettes, kann man diese durch Gehen über das Brett passieren. Sobald das Ziel erreicht wurde, entzündet sich das Warnfeuer automatisch, das Spiel ist gewonnen. Das Spiel umfasst drei Level mit sich steigerndem Schwierigkeitsgrad. Für jedes dieser Level ist eine bestimmte Maximaldauer festgelegt, wenn der Timer diese überschreitet gilt das Level automatisch als verloren. Fällt man von einer Brücke und berührt die Wasseroberfläche, hat man ebenfalls verloren.



Abbildung 7: Textausgabe Sieg



Abbildung 8: Textausgabe Verlieren

## 5 Debug Options

### 5.1 Adjustable Parameters

Die erforderlichen Adjustable Parameters werden im Config-File 'settings.ini' gesetzt und das Spiel berücksichtigt diese entsprechend. Wenn das Spiel in einem anderen Level als Level 1 gestartet werden soll, kann dies auch im 'settings.ini'-File festgelegt werden. Für

die Effekte oder Einstellungen, die mit Tastendruck verändert werden können, erfolgt bei Veränderung eine entsprechende Ausgabe in der Konsole.

## 5.2 View-Frustum-Culling

Für das View-Frustum-Culling wird für jedes Objekt eine Bounding-Sphere erstellt und überprüft, ob Teile der Sphere im sichtbaren Bereich sind. Wenn das zutrifft wird das umschlossene Objekt gezeichnet.

Quellen: <http://www.crownandcutlass.com/features/technicaldetails/frustum.html>

## 6 Verwendete externe Bibliotheken

- FreeImage-Library zum Laden von Bildern;  
Quelle: <http://freeimage.sourceforge.net/download.html>
- NVIDIA PhysX, Physics-Library für Collision Detection; Quelle: NVIDIA GameWorks, Download über GitHub
- Freetype-Library zum Laden von Fonts; Quelle: <https://www.freetype.org>
- Assimp Model-Loading-Library Version 4.1.0;  
Quelle: <http://www.assimp.org/index.php/downloads>