

Submission Runner

Viktoria Pundy
Verena Widhalm

Story

Oh nein! Du hast schon wieder viel zu spät mit der Aufgabe begonnen, bist gerade noch fertig geworden und nun droht die Deadline dich zu überrollen und Deine Abgabe nicht mehr anzunehmen. Hilf Deiner Abgabe, rechtzeitig beim Abgabeserver anzukommen und schließe damit den Kurs ab! Steuere sie durch die tückischen Leitungen des Internets und weiche ihren Hindernissen aus. Lass Dich jedoch dabei nicht von der Deadline hinter Dir und den Dich behindernden Bugs erwischen! Zum Glück helfen dir die Tutoren am Weg dein Ziel schneller zu erreichen. Du hast den Kurs erfolgreich bestanden, wenn alle Abgaben rechtzeitig beim Server angekommen sind.

Gameplay

Ziel dieses 3D Spiels ist es, die Figur so schnell wie möglich durch den Hindernisparcour zu steuern und zum Ziel zu gelangen. Als Zeitbegrenzung befindet sich hinter dem Spieler eine Wand, die Deadline, welche diesen nicht einholen darf. Weiters befinden sich im Spiel verschiedene Hindernisse, welche den Spieler lähmen können, und Boni welche dem Spieler einen zusätzlichen Boost verschaffen. Hat der Spieler das Ziel erreicht, kann er zum nächsten Level teleportiert werden. Sobald das Ziel vom letzten Level erreicht wird, ist das Spiel gewonnen.

Anmerkung: Unser zweiter Level ist derzeit noch in Bearbeitung.

Boni

Die Boni unterstützen den Spieler bei seiner Mission. Das Objekt schwebt leicht bewegend in der Luft. In der Mitte des Boni befindet sich ein drehendes Bild. Sobald der Spieler einen Boni erreicht, erhält dieser eine kurze Beschleunigung und fällt danach wieder in seine vorherige Geschwindigkeit zurück. Weiters wird für kurze Zeit eine kleine Nachricht im linken unteren Bereich angezeigt.

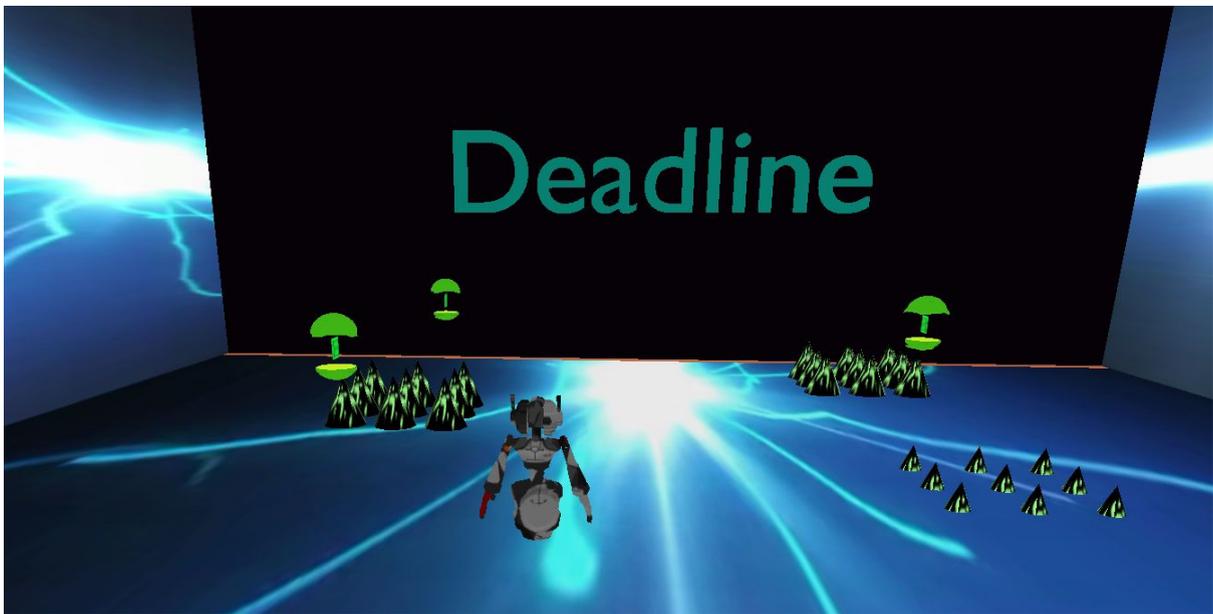
Bug

Die "Bugs" sind Gegner des Spielers. Sie schweben zu Beginn in der Luft. Wenn die Distanz zwischen einem Spieler und einem "Bug" klein genug ist, fällt der "Bug" auf den Boden und verfolgt den Spieler. Wenn die Distanz zwischen den beiden einen Schwellenwert übertritt,

verfolgt der "Bug" den Spieler nicht mehr und bleibt auf seiner derzeitigen Position. Wenn dieser Schwellenwert erneut unterboten wird, beginnt er von Neuem, den Spieler zu verfolgen. Verfolgt der Bug den Spieler, so leuchtet er auf und beleuchtet dadurch seine Umgebung mit einem Pointlight. Man kann die Bugs besiegen, indem man mit Bullets auf sie schießt. Trifft ein Bullet einen Bug, so zerfällt dieser und kann keinen Schaden mehr anrichten.

Wird der Spieler von einem Bug getroffen, erscheint weiters eine kleine Nachricht im linken unteren Bereich.

Deadline



Die Deadline ist eine Wand, welche den Spieler verfolgt. Kollidiert der Spieler mit ihr, so ist die Runde vorbei und der Spieler kann entscheiden, ob er den Level oder das ganze Spiel neu starten möchte. Befindet sich die Wand zwischen Spieler und Kamera, so wird sie als Linie am Boden angezeigt. Ansonsten wird sie als Wand mit den Worten "DEADLINE" dargestellt.

Water

Das Water Mesh besteht aus einer Plane und wurde mittels Tessellation shader unterteilt und anschließend animiert.

Finishline

Die Finishline befindet sich am Ende jedes Levels und wird als weiße Fläche dargestellt. Sobald der Spieler den Trigger der Finishline lebend erreicht, wird eine Zielnachricht angezeigt und man kann in das nächste Level wechseln.

Bullet

Die Bullets sind die Verteidigungsmittel des Spielers. Er kann sie einsetzen, um Bugs zu zerstören. Bullets werden vom Spieler aus geschossen und wandern eine Zeit lang durch den Level. Sie sind als "1" und "0" dargestellt.

Compulsory Effects

- **HUD:** Das Heads-Up Display wurde mithilfe der Library Freetype (<https://www.freetype.org/>) implementiert. Zum Verständnis der Funktionsweise wurden die Dokumentation der Freetype Homepage (<https://www.freetype.org/freetype2/docs/tutorial/index.html>) sowie das Tutorial von LearnOpenGL (<https://learnopengl.com/In-Practice/Text-Rendering>) zum Verständnis herangezogen. In unserem HUD werden primär zwei Elemente angezeigt: Einerseits Text, welcher zu unterschiedlichen Zwecken an verschiedenen Stellen auf dem Bildschirm angezeigt wird, andererseits ein Quadrat, welches die Distanz zwischen dem Spieler und der Deadline darstellt.

Das HUD ist eine eigene Klasse, welche bei Start des Spiels aufgesetzt wird. Dazu werden zwei VAOs und zwei VBOs initialisiert: Ein VAO/VBO für Text und ein VAO/VBO für das Quadrat. Danach werden die ersten 128 Elemente des Ascii-Codes einzeln geladen und als Textur gespeichert.

Für das Quadrat welche den Abstand von Spieler zu Deadline darstellt, wird In der Render-loop in jedem Durchlauf die Distanz zwischen Spieler und Deadline berechnet. Anhand dieser wird das angepasste Quadrat gezeichnet:

- maximale Distanz, die der Balken anzeigen kann: 100 Einheiten
- gezeichnete Höhe: Minimum von 100 und der Distanz
- Balkenfarbe: die Farbe des Balkens wird durch seine Höhe bestimmt

Je kleiner der Balken, desto höher der rote Farbanteil, je größer, desto höher der grüne Farbanteil. Für den Balken gibt es dabei eine untere Grenze, damit ein Teil des Balkens jederzeit sichtbar ist. Die berechneten Eckpunkte werden im VBO abgespeichert und gezeichnet.

Für den Text existiert eine Methode, welche über den gegebenen Text iteriert und sich anhand von dessen Eigenschaften ausrechnet, an welcher Position welcher Buchstabe stehen sollte. Für jeden Buchstaben werden erneut die Eckpunkte im VBO abgespeichert und gezeichnet.

Die einzelnen Texte können bei Bedarf an die Methoden des HUD übergeben werden. Weiters kann das HUD umgeschaltet werden: Mit Shift kann das HUD versteckt oder (erneut) angezeigt werden.

- **Light mapping:** Unser Spiel beinhaltet pro Level eine statische Lichtquelle sowie die Lichtquellen der Bugs. Die statische Lichtquelle befindet sich am Ende des Levels

und erleuchtet das Ziel. Diese Lichtquelle wird mittels Lightmapping auf den statischen Objekten simuliert. Dazu wurden die statischen Objekte der Szene in Blender nachgebaut, die Lichtquelle platziert und die Lightmaps wie im Tutorial gezeigt erstellt. Die daraus erstellten Bilder dienen als Textur.

Da die Wände sowie Boden und Decke unseres Levels eine Video-Textur besitzen, wurde die Lichtinformation für diese Objekte als eine zusätzliche Textur erstellt und die beiden Texturen werden im Programm kombiniert.

- **Physics Engine:** Hier haben wir uns für NVidia PhysX entschieden. In unserem Spiel wurden sowohl Collider als auch Trigger verwendet. Weiters lassen sich die Objekte in statische und dynamische unterteilen. Statische Objekte können mit den Dynamischen kollidieren ohne diese zu durchschreiten, z.B. Bug wird durch Wand aufgehalten.

Die verwendete Gravität wurde mit 9.81 festgelegt bzw. für die fliegenden Boni auf 0 gesetzt.

Effects with Effect Points

- **Water Mesh (Verena Widhalm):**

Für das Watermesh wurden zuerst einfache Planes mit jeweils 4 Vertices erstellt und anschließend mit Tessellation shader unterteilt. Die Berechnung der Wellenbewegung erfolgt durch aneinanderreihung von Sinus und Cosinus Berechnungen. Die Normalen wurden wie im Tutorial erwähnt mittels ableiten neu berechnet. Zum erstellen dieses Effekts wurden folgende Quellen verwendet:

- Textur: <http://www.cadhatch.com/seamless-water-textures/4588167784>
- Tessellation:
 - http://www.rendering.ovgu.de/rendering_media/downloads/fohlen/gpurog/05_geometryshader
 - <http://prideout.net/blog/?p=48>
 - <http://codeflow.org/entries/2010/nov/07/opengl-4-tessellation/>
- Wellenberechnung:
 - https://ac.els-cdn.com/S1877705812002299/1-s2.0-S1877705812002299-main.pdf?_tid=4e43bc4f-8a21-4d0b-9683-9ed495f12486&acdnat=1528881875_191efc10a99fcb00f0677e3c2baf283
 - <https://jayconrod.com/posts/34/water-simulation-in-glsl>

- **Particle System (Verena Widhalm):**

Das Particle System wurde anhand des Tutorials der Vorlesung erstellt.

(https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/Particles_SS17.pdf). Weitere Tutorials waren daher nicht nötig. Das Particle System wird

ausgelöst, sobald ein Bug durch eine Bullet vernichtet wird. Es werden jeweils 50 Particle Positionen mit random Geschwindigkeitsvektoren erstellt und mittels SSBO an VAO gebunden. Die Partikle verschwinden, sobald ihre Lebensdauer zu ende ist.

- **kd-Tree for culling (Viktoria Pundy):** siehe "View Frustum Culling"
Der KD-Tree wurde von den Folien der Parallel-Vorlesung Computergraphik VO

abgeleitet: https://www.cg.tuwien.ac.at/courses/CG/slides/DataStructures_Slides.pdf, Folie 52ff. Tutorials wurden keine verwendet, da keine gefunden werden konnten.

- **Video textures (Viktoria Pundy):** Für die Anwendung von Video Textures wurde die Library OpenCV verwendet (<https://opencv.org/>). Mithilfe dieser Library wird das Video zu Beginn eingelesen und geöffnet. In der Render-Schleife wird alle 0.05 Sekunden der nächste Frame des Videos ausgelesen und temporär abgespeichert. Danach wird dieser Frame für die gewünschte Textur gebunden und gespeichert. Ist das Video am Ende angelangt, werden die Frames von Vorne ausgelesen. Die Texturen werden dementsprechend in Echtzeit aktualisiert. Zu Beginn war geplant, die Video Textur für unsere Gegner-Objekte, die Bugs, einzusetzen. Durch das komplexere Modell dieser ermöglicht die Video Textur jedoch keinen schönen Effekt und ist fast nicht erkennbar. Deswegen verwenden wir die Video Textur nun an den Wänden, der Decke und dem Boden. Das Video zeigt elektrische Blitze, welche sich immer wieder ändern. Der Effekt ist dementsprechend sofort zu Beginn des Spiels sichtbar.

Die Funktionen von OpenCV wurden mithilfe zweier Foreneinträge (<http://answers.opencv.org/question/32747/opencv-move-to-begin-frame-position-fail/> und https://stackoverflow.com/questions/33503138/how-to-extract-video-frames-and-save-them-as-images-using-c?utm_medium=organic&utm_source=google_rich_ga&utm_campaign=google_rich_ga) implementiert.

Complex 3D Models from Files

Um komplexe Objekte in das Spiel laden zu können, wird die Library assimp (<http://www.assimp.org/>) verwendet. Um die Daten aus den Files lesen zu können, wurde die Klasse ModelLoader erstellt. Zu Beginn wird überprüft, ob das Modell bereits geladen wurde. Ist dem nicht so, wird das Modell neu geladen. Dazu wird die Szene aus einem .obj File mit Assimp importiert und temporär gespeichert. Danach wird eine rekursive Methode (welche mit dem Wurzelknoten zu Beginn aufgerufen wird) aufgerufen, in welcher jedes Mesh des Knotens verarbeitet wird. Von jedem einzelnen Mesh werden Position, Normalen, UV-Koordinaten, Indices und Texturen gespeichert. Weiters werden mit derselben Methode die Kinderknoten aufgerufen. Das geladene Modell wird abgespeichert und kann danach jederzeit wieder aufgerufen werden. Dadurch ist eine geringere Ladezeit während der Render-Schleife möglich.

Es wurden sowohl Modelle aus dem Internet als auch mit Blender selbst erstellte Modelle geladen. Die Modelle der Bugs (Gegner, <https://www.cgtrader.com/free-3d-models/space/other/air-glided-system-backpack>) und des Spielers (<https://www.turbosquid.com/FullPreview/Index.cfm/ID/1068325>) wurden aus dem Internet geladen. Die Modelle der Bonis und Bullets ("1"er und "0"er) wurden selbst erstellt.

Animation Using Physics Engine

Durch die Gravity bzw. deaktivierung der Gravity werden die einzelnen Objekte in der Szene beeinflusst. Bugs und der Spieler können Springen und auch mit Wänden kollidieren. Bonis haben keine Gravity und können in der Luft schweben.

Hierarchical Animation (manually)

Für die hierarchische Animation haben wir das Boni Objekt verwendet. Dieses besteht aus den Komponenten BoniHull und BoniCenter. Die BoniHull bewegt sich leicht nach oben und unten. Das BoniCenter befindet sich in der Mitte des BoniHull und bewegt sich mit diesem mit, zusätzlich rotiert es auch noch.

Frustum Culling

Einer unserer Effekte ist "kd-Tree for Culling", weswegen für die Technik des View Frustum Cullings ein KD-Tree verwendet wurde. Da von den einzelnen Level je nur die Objekte des aktiven Level geladen werden, wurde der Bereich des KD-Trees ungefähr auf den Bereich, der den Level umfasst, angepasst.

Für den aktiven Level werden zwei KD-Trees geladen: Der erste beinhaltet die statischen Elemente der Szene. Da sich die Positionen dieser Elemente nicht verändern können, muss dieser Baum nicht aktualisiert werden und nur während des Ladens des Levels aufgebaut werden. Der zweite KD-Tree beinhaltet die dynamischen Elemente. Da sich die Positionen der Elemente verändern können, muss auch der KD-Tree aktualisiert werden. Da sich innerhalb eines Frames die Position der Objekte nur geringfügig ändert, wird diese Aktualisierung alle 0.1 Sekunden ausgeführt. Dazu wird der Tree vollkommen geleert und die Objekte neu eingefügt.

Die Knoten der KD-Trees beinhalten je einen Teil des Level-Bereichs. Dabei ist jeder Knoten innerhalb des Bereichs seines Parent-Knoten. Aus Performance-Gründen wird jeder Bereich halbiert und die beiden Kinder repräsentieren je eine Hälfte des Bereichs. Die Wahl, ob der Bereich entlang der x-, y- oder z-Achse geteilt wird, erfolgt anhand der Größe der jeweiligen Dimensionen dieses Bereichs: die größte Dimension wird geteilt. Hat beispielsweise ein Bereich eine Breite von 50, eine Höhe von 60 und eine Tiefe von 100, so werden zwei Unterbereiche mit einer Breite von 50, einer Höhe von 60 und einer Tiefe von 50 erstellt. Bei jedem Knoten wird eine Liste der Objekte, die sich (teilweise) in dessen Bereich befinden, abgespeichert.

Jedes Objekt, welches in den KD-Tree eingefügt werden soll, besitzt eine Object-Boundingbox. Für jeden Eckpunkt wird rekursiv kontrolliert, in welchen Bereichen der Knoten des Baums er sich befindet und gegebenenfalls zu dessen Objektliste hinzugefügt. Dies wird für alle Objekte durchgeführt. Erwähnt sei hierbei, dass nur Knoten, in denen sich mindestens ein Eckpunkt eines Objekts befindet, angelegt werden.

In der Render-loop wird jeden Frame rekursiv überprüft, welche KD-Tree-Knoten im View-Frustum liegen. Liegt der Bereich eines Knoten vollständig außerhalb des Frustum, wird die Überprüfung für den Knoten abgebrochen und die weiteren (noch nicht geprüften) Geschwisterknoten überprüft. Liegt ein Knoten vollkommen im Frustum, werden die Objekte dieses Knotens sofort gezeichnet, die Kinder-Knoten müssen nicht mehr überprüft werden. Liegt der Bereich teilweise innerhalb des Frustum, werden die Kinderknoten auf dieselbe Art und Weise überprüft.

Der Effekt ist besonders zu Beginn des ersten Levels erkennbar: Wenn der Frame-Time Output aktiv ist, wird rechts oben an der dritten Stelle die Anzahl der gezeichneten Objekte angezeigt. Wird die Kamera nun nach links oder rechts gedreht, sollte sich diese Anzahl stark senken und die Frames/sec (an erster Stelle) deutlich erhöhen.

Controls

Kurzzusammenfassung:

Taste	Effekt
A, W, S, D	Steuerung der Figur
Maus	Kontrolle der Kamera
X	Schießen
Leertaste	Springen
Shift	Togglen des HUD
ESC	Beenden des Spiels
8	God-Mode
F2	Anzeigen von Frame Information
F3	Wireframe Mode aktivieren/deaktivieren
R	Neustart des Spiels ¹
L	Neustart des Levels ¹
T	Wechsel zu nächstem Level ²

¹ Nur nach Berührung mit der Deadline möglich

² Nur nach Erreichen des Ziels möglich

Beschreibung

Der Spieler kann die Figur mit den Tasten "W", "A", "S" und "D" gesteuert werden. Die Bewegung der Figur kann durch Wände und andere solide Hindernisse verhindert werden. Die Kamera kann mit der Maus bewegt werden. Die Bewegung wird jedoch eingeschränkt, falls sie dazu führen würde, dass die Kamera hinter einer Wand und damit außerhalb des Level ist. Weiters kann mit dem Mausrad gezoomt werden.

Um die Gegner, die Bugs, besiegen zu können, hat der Spieler die Möglichkeit, mit X zu schießen. Wird ein Bug von einer Patrone getroffen, so ist dieser zerstört und kann keinen Schaden mehr anrichten. Der Spieler kann mit der Leertaste springen. Dabei sei anzumerken, dass diese Fähigkeit nur ausgeführt werden kann, wenn sich der Spieler am Boden befindet. Doppelsprünge sind damit nicht möglich.

Das Heads-Up Display kann sowohl mit Left-Shift als auch Rechts-Shift aktiviert oder deaktiviert werden. Standardmäßig ist es aktiv. Wird das HUD deaktiviert, so werden keine textuellen Informationen ausgegeben und der Balken, der die Distanz zur Deadline angibt, wird ausgeblendet.

Mit "8" kann der God-Mode aktiviert und deaktiviert werden. Der God-Mode deaktiviert Reaktionen auf Boni und Bugs sowie Stacheln. Weiters hat die Berührung mit der Deadline keine Auswirkung. Zusätzlich kann die Kamera frei bewegt werden.

Wenn der Spieler das Ziel eines Levels erreicht hat, wird eine Nachricht am Bildschirm angezeigt. Drückt der Spieler "T" so wird er an den Anfang des nächsten Level gebracht. Wird der Spieler hingegen von der Deadline eingeholt, so kann er "L" (um den Level neu zu starten) oder "R" (um das Spiel vollkommen von Neuem zu beginnen) drücken.

Debug Options

Für jede Aktivierung/Deaktivierung eines Debug Outputs wird im Fenster links oben angezeigt, welcher Output aktiviert/deaktiviert wurde.

- **F2 - Frame Rate**

Die Information zur Frame Rate werden mithilfe der Library Freetype angezeigt. Mit F2 kann der Output angezeigt/ausgeblendet werden. Werden die Informationen angezeigt, so werden rechts oben drei Zahlen eingeblendet: Die oberste Zahl repräsentiert die Frames pro Sekunde. Diese Zahl wird dementsprechend jede Sekunde aktualisiert. In der zweiten Zeile wird in Millisekunden die Zeit für den Durchlauf eines einzelnen Frames angezeigt. In der dritten Zeile wird die Anzahl der gezeichneten Objekte angezeigt.

- **F3 - Wire Frame**

Mit F3 kann der Wire-Frame Modus angezeigt werden. Um dies zu ermöglichen wird `glPolygonMode` aufgerufen und bei Aktivieren des Modus mit `GL_Line` und bei Deaktivierung mit `GL_FILL` aufgerufen. Die Technik wurde aus dem ECG-Framework übernommen.

- **F8 View-Frustum Culling**

Der Effekt View-Frustum Culling kann mithilfe von F8 deaktiviert/aktiviert werden. Dazu wird in der Render-Schleife überprüft, ob View-Frustum Culling aktiv oder inaktiv ist. Ist es aktiv, wird mittels KD-Tree überprüft, ob ein Objekt gezeichnet werden sollte. Ist es deaktiviert, werden alle Objekte ohne Überprüfung gezeichnet. Durch Deaktivierung des View-Frustum Cullings sinkt die Framerate sichtbar.