

Documentation Tower Delivery

Introduction

Tower Delivery ist ein Jump'n'Run Spiel bei dem es darum geht, auf einen Turm zu klettern und dabei Pakete zu sammeln. Zum Hinaufklettern muss man von Container zu Container springen und teilweise Cubes verschieben, um sich selbst einen Weg nach oben zu bauen.

Unser Framework basiert auf dem Hazel Engine Serie von TheCherno (<https://github.com/TheCherno/Hazel>) und ist in die zwei Solutions "TowerDelivery" und "Game" unterteilt. Aus dem "TowerDelivery" Projekt wird eine static Library erstellt, die in "Game" geladen wird und dort als Engine dient.

Als Build-System verwenden wir Premake. Über "generateProjects.bat" lassen sich die .sln-Dateien erstellen.

Gameplay

Wir haben alle Punkte aus der Compulsory Kategorie implementiert. Weiters sind folgende Punkte in unserem Spiel vorhanden:

- Collision Detection (Basic Physics)
- Advanced Physics (6 Points): Im Level finden sich dynamische Würfel, die vom Player bewegt werden können. Außerdem verwenden wir Raycasting, um zu überprüfen, ob der Player am Boden ist, bevor er springt.
- Heads-Up Display (4 Points)

Außerdem haben wir ein Checkpoint System implementiert, das den Player an den höchsten erreichten Checkpoint zurücksetzt, sollte er vom Turm fallen und noch Leben übrig haben. Dabei werden zusätzlich die Positionen der verschiebbaren Würfel zurückgesetzt.

Effects

Particle System (CPU)

Wir haben ein Particle System (CPU) implementiert, wobei wir uns am verlinkten Tutorial orientiert haben. Als Particle rendern wir eine kleine Box. Die Particle Systems sind über das ganze Level verteilt und dienen als Check-Point/Respawn-Point.

<http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/>

Specular Map

Eine Specular Map wird genutzt, um die bewegbaren Cubes zu rendern und ihnen eine glänzende Textur zu geben. Implementierung von learnOpenGL.

<https://learnopengl.com/Lighting/Lighting-maps>

Physically Based Shading

Für die Container und den Player Character nutzen wir einen PBR shader, dem fünf Texturen (albedo, normal, metallic, roughness, ao), die Point Light Positionen und Farben und die Kameraposition als uniform übergeben werden.

<https://learnopengl.com/PBR/Theory>

<https://learnopengl.com/PBR/Lighting>

Bloom

Bloom wird in unserem Level verwendet, um "Lichter" zu rendern. Dabei zeichnen wir einen Cube an die gleichen Stellen wie die Point Lights und wenden dann den Bloom Effekt auf die besonders hellen Stellen des Bilds an.

<https://learnopengl.com/Advanced-Lighting/Bloom>

Keybinds

Action	Keybind
Charakter bewegen	W, A, S, D
springen	Spacebar
toggle zwischen 3rd Person Kamera und steuerbarer Kamera	F2
bewegbare Kamer auf (x,z)-Ebene steuern	Pfeil-Tasten
bewegbare Kamera auf y-Achse steuern	right SHIFT, right strg
toggle zwischen HUD und kein HUD	F3
Fenster schließen	ESC

Additional Libraries

Assimp

Wir nutzen assimp um .obj Files, die in blender erzeugt wurden, als Models in unser Game zu laden. Assimp ist als eine static library als submodule geaddet und per premake inkludiert und verlinkt.

Quelle: <https://github.com/AminAliari/assimp>

Bullet

Bullet ist die Physics-Engine die in unserem Game für Collision Detection und physikbasierte

Bewegungen (springen und Cubes verschieben) genutzt wird. Beim Inkludieren von bullet haben wir uns am CG UE-Tutorial orientiert und dieses mittels premake umgesetzt.

Quelle: <https://github.com/bulletphysics/bullet3>

freeType

Zum laden und rendern von Text als unser HUD verwenden wir FreeType. FreeType ist als static library included.

Quelle: <https://github.com/ubawurinna/freetype-windows-binaries>

Glad

Als OpenGL Loading Library nutzen wir Glad. Mit premake legen wir die Include-Directories und Links fest.

Quelle: <https://github.com/Dav1dde/glad>

GLFW

GLFW benutzen wir für Window management. Eingebunden ist es über einen Fork von TheCherno (<https://github.com/TheCherno/glfw>), der ein passendes premake-file liefert.

Quelle: <https://github.com/glfw/glfw>

GLM

GLM verwenden wir für einfache mathematische Funktionen und Klassen wie Vektoren und Matrizen. Es ist über ein subproject inkludiert.

Quelle: <https://github.com/g-truc/glm>

inih

Um Einstellungen aus dem settings.ini-File zu lesen verwenden wir inih.

Quelle: <https://github.com/benhoyt/inih>

spdlog

Spdlog hilft bei der Ausgabe auf der Konsole dabei, verschiedene Formatierungen (v.a. Farben) anzuwenden. Die Library ist über ein subproject inkludiert.

Quelle: <https://github.com/gabime/spdlog>

stb_image.h

Stb_image nutzen wir zum Laden von Texturen. Die Library ist als Header-File inkludiert.

Quelle: https://github.com/nothings/stb/blob/master/stb_image.h

Sources Texturen

Character: <https://freepbr.com/materials/metal-plates-panel/> (PBR Shader)

Container: <https://freepbr.com/materials/chipped-paint-metal/> (PBR Shader) → Farben von uns angepasst

Cubes: <https://freepbr.com/materials/storage-container2/> (nur albedo und metallic/specular → Specular Map)

Floor: <https://freepbr.com/materials/concrete-2/> (nur albedo und metallic/specular → Specular Map)

Particles: https://www.freepik.com/free-vector/various-cardboard-boxes-flat-icon-set_9176967.htm#page=1&query=cartoon%20box&position=1 Designed by pch.vector / Freepik

Anm.: Models für Container und Player-Character sind von uns selbst auf blender modelliert worden.