

9.806

Abgabe 3 alleine:

Mein Kollege hat sich leider nicht mehr gemeldet. Somit war ich für die Aufgabe 3 komplett auf mich allein gestellt.

Lernen:

Das Einarbeiten und Erlernen der Thematik wurde Mithilfe des Buches „Introduction to 3D Game Programming with DirectX10, Frank D. Luna, worldware game and graphics library“ gemacht.

Implementierung

Gameplay:

Das Gameplay kann: Zu Beginn sucht man sich im Menü eines der 3 Levels(LEVEL 1 ist am ausführlichsten implementiert, Level 2 und 3 lege ich wenn ich Zeit hab noch für die Präsentation nach) aus oder Quit für Beenden.

Im Level selbst, SPACE drücken und der Avatar fliegt los. Mit „WASD“ geht's dann ans Lenken und Beschleunigen(„W“ gedrückt halten um zu Beschleunigen).

ZIEL: Ganz nah an der Wand fliegen. Das bringt viele Punkte(links oben eingeblendet)
Links oben eingeblendet sind auch die aktuellen Rekordpunkte für dieses Level. Bei Überbieten dieser Punkte(dazu muss man sicher mit dem Fallschirm landen(SPACE)) werden die aktuellen Punkte für den nächsten Flug vorgemerkt. Landet man(mit oder ohne Fallschirm), ladet wieder das Spielmenü.

Umsetzung des Gameplay:

- Game State:
->`void Game::updateScene(float dt)`
Die Variable „game_state“ gibt den aktuellen Status im Spiel an(Menü, Stehen, fliegen, springen)
- ->Menü: `Game::menu(float dt)`
Die 4 Menüpunkte sind simple Quader mit Texturen drauf. Die State-Variable „menu_pos“(Steuerbar über „W“ und „S“). gibt die aktuelle Menüposition an. Anhand dieser wird auch entschieden welche Textur auf den Menüquaden geladen wird.
- `Game::stand(float dt)`: Avatar steht
- `Game::jump(float dt)`: Avatar springt los
- `Game::fly(float dt)`:
 - Hauptsteuerung über Verändern der Rotations und Translationsmatrizen(„mAvatarWorldR“, „mAvatarWorldT“)
 - Kollisionserkennung über „Distance::instance()->getMinDistance()“ berechnet die minamlste Distanz zwischen allen Levelpolygonen und dem Avatarzentrumspunkt.
Quelle: <http://mathworld.wolfram.com/Point-PlaneDistance.html>
Da diese Berechnung auch über die Polygongrenzen hinausgeht(Ebene) hab ich in zusätzlich vorher für die Auswahl der richtigen Polygone die 3 Dreieckswinkel zwischen Polygonecken und Avatar berechnet („Distance.cpp“ Zeile 226-238)

Nichttriviale Objekte

Fallschirm(SPACE drücken)
Level ist auch konvex

Levelerzeugung und Avatarerzeugung

1. Blender: .blend File (Level bestehende aus Polygonen) Bsp: material/plane1.blend
2. Blender: .blend File → .x File
3. .x File wird nach Auswahl in Menü geladen mit „Game::loadComplexObject“
Ist im Prinzip ein Parser der die Daten aus dem .x File für das DirectX rendern aufzubereiten.

Animierte Objekte

Avatar mit Fallschirm (hierarchische Animation ist leider noch nicht drin)

Beschleunigung der Sichtbarkeitsberechnung

Camera::perspective(float fovy, float aspect, float znear, float zfar)
-znear und zfar geben den Sichtbarkeitsbereich der Kamera an(near and far plane)

Shadow Maps

„shadow.fx“: Schatten wird mit Hilfe von CalcShadowFactor berechnet. Dies dient dann „lighthelper.fx“ - „float3 ParallelLight“ den möglichen Grauwert hinzuzunehmen.

Statische Levels of Detail für Modelle

Es gibt zwei Modelle des Avatar.

1. Avatar ohne Hände/Füße
2. Avatar mit Hände/Füße

Über eine Distanzberechnung von Kamera zu Avatar wird entscheiden welcher Avatar gerendert wird(„Distance::instance()->calcLod“).

Steuerung

SPACE

- Sprung von Klippe
- Fallschirm öffnen

'W'

- vorwaerts in Kopfrichtung fliegen
- Menu nach oben

'S'

- Menu nach unten

'A'

- nach links drehen

'D'

- nach rechts drehen

Kamera:

- Linke Maustaste gedrückt halten