

## CG DOKUMENTATION - TASK 2

### Bewegung in Crystals

- Mit W,A,S,D und SPACE bewegt man sich durch den Raum
- Mit F interagiert man mit Objekten
  - Durch das Mesh Cooking kommt es vor allem bei der Interaktion mit den Crystal Objekten zu Problemen. Am besten man versucht auf die Crystals darauf zu springen und von Oben nach unten mit F versuchen zu interagieren. Meistens funktioniert das nach ein paar mal drücken
  - Interagiert man mit einem Objekt so gibt es dazu auf der Konsole einen Output. Benutzt man einen Key Crystal in ein Key Logic Unit Gate (oben Gelb - steht meistens neben einer Tür), so unlocked diese die Tür daneben, wenn man danach mit der Tür interagiert, gelangt man ins nächste Level (momentan gibt es zwei).
- Mit F3 gelangt man in den God-Mode und kann so durch das Level schweben. Von da aus gibt es wieder zwei Möglichkeiten
  - Drückt man erneut F3 an einer beliebigen Stelle, so kommt man wieder zu der ursprünglichen Position zurück.
  - Drückt man stattdessen F4, so wird der Character an der momentanen Stelle gespawnd und man kann von dieser Position normal weiterspielen.
- Mit F5 Kann man sich diverse Statistiken auf dem Bildschirm darstellen lassen
- Mit F6 shaltet man das view frustum culling ein/aus.

### Beschreibung der Implementierung

- **Freely movable camera:** Für die Kamera wurde ein neuer header "ownCamera.h" und ein neues .cpp file "ownCamera.cpp" erstellt. Der header besteht aus einigen notwendigen Variablen, welche zum Beispiel die Position oder die view Matrix enthalten, sowie mehrere Methoden, unter anderem die wohl wichtigste "update" Methode. Im .cpp file werden die Methoden ausdefiniert. Sogut wie alle Methoden sind entweder Getter oder Setter, abgesehen von der "update" Methode. Diese hat 7 Parameter: **float dt, bool forwards, bool backwards, bool left, bool right, float yaw, float pitch.**
  - CumDt = delta time, für die Frame Unabhängigkeit. x
  - forwards = true, falls der W Button betätigt wurde. Teilt der Methode mit, dass der User sich nach vorne bewegen will.
  - backwards = true, falls der S Button betätigt wurde. Teilt der Methode mit, dass der User sich nach hinten bewegen will.



- left = true, falls der A Button betätigt wurde. Teilt der Methode mit, dass der User sich nach links bewegen will.
- right = true, falls der D Button betätigt wurde. Teilt der Methode mit, dass der User sich nach rechts bewegen will.
- yaw, pitch = 2 der 3 eulerschen Rotationswinkel. Werden verwendet, damit der User nach oben, unten, links und rechts schauen kann.

In der ersten Zeile wird einer Variable "speed" das Produkt von der im header vordefinierten "cameraSpeed" Variable und der dt zugewiesen. Für die Bewegung nach vorne bzw. nach hinten wird das Produkt von "speed" und der "\_cameraFront" Variable, welche den Richtungsvektor enthält von der Richtung in welche die Kamera schaut (default = (0.0f, 0.0f, -1.0f), zur/von der aktuellen Position addiert bzw. subtrahiert.

Die links/rechts Bewegung hingegen ist ein wenig komplizierter. Hierbei multipliziert man aber "speed" mit dem normalisierten Kreuzprodukt von dem "\_cameraFront" Vektor und dem "\_cameraUp" Vektor und addiert bzw. subtrahiert dieses Produkt zur/von der aktuellen Kamera Position. Der "\_cameraUp" Vektor ist hierbei der Vektor, welcher von der Kamera aus nach oben zeigt.

Anschließend wird die Drehung anhand von 2 Rotationsvariablen simuliert. Dabei wird ein Hilfs-Vektor erstellt und die drei Positionen x,y und z anhand der eulerschen Rotationsgesetze belegt. Dabei werden die beiden Werte yaw und pitch zuerst zu Radianten formatiert und anschließend werden die cosinus und/oder sinus Werte von den Radianten miteinander multipliziert. Der nun entstandene Vektor enthält alle Details der Rotation und wird anschließend normalisiert der "\_cameraFront" Variabel zugewiesen.

Die yaw und pitch Werte werden in der Main Klasse in der Methode "mouse\_callback" ausgerechnet, da diese von der Position des Mauszeigers abhängen. Am Anfang der Methode wird geschaut, ob das der erste Aufruf dieser Methode war, sprich ob das Programm gerade gestartet hat. Ohne dieser Überprüfung würde die Kamera beim starten einen Sprung machen, da die Methode mit den x und y Werten der Maus ausgeführt worden wäre, als die Maus den Bildschirm "betreten" hat. Danach wird die Abweichung von den vorherigen Koordinaten und den aktuellen berechnet und dieser Offset multipliziert mit der festgelegten Drehgeschwindigkeit zu yaw im Falle der x Koordinate und pitch im Falle der y Koordinate addiert. Abschließend wird noch überprüft, ob errechnete pitch Wert zulässig ist und



gegebenenfalls angepasst. Das führt dazu, dass der User nie über 90° bzw unter -90° schauen kann.

Mit all den notwendigen Parameter wird nun die Viewmatrix durch die lookAt Funktion von glm erstellt und anschließend mit der Projektionsmatrix multipliziert, um die Viewprojectionmatrix zu erhalten.

- **Moving Objects:**

- Es gibt neben den Key - Logic Unit Gates (kurz LUT) noch andere LUTs (Functional LUTs) welche gewisse Objekte triggern und eine Animation auslösen. Diese Animationen können, je nach Objekt mit einen Loop ausgestattet sein oder müssen immer wieder aufs neue getriggert werden. Solche Verhaltensweisen wird durch die umfangreiche Game Logik übernommen.
- Abgesehen davon kann es auch ganz unabhängig davon animierte (meistens mit loop)

- **Texture Mapping:**

- Die Objekte, welche mit Assimp importiert werden haben spezielle Namen: Meistens sehen diese Namen so aus: "id(x):\*BeliebigeZeichen\*". x..steht dann für eine Objekt-ID. Zu jedem Level gibt es zusätzlich eine XLevel.ini (X steht für die Level Zahl) in welchem pro Objekt (per ID) seine Eigenschaften ausgelesen werden können. Unter anderem kann es für ein Objekt eine Variable "baked" geben. Wenn diese true ist, so wird auf das Objekt eine Textur "levelX\_x.dds" (X..Level, x..ObjID) gemapped. Diese Texturen sind natürlich mit Lichtinformationen gebaked worden.
- Einige Objekte (z.B. LUTs) haben immer die gleiche gebakete Textur.

- **Simple lights and materials:**

- Beleuchtungen werden als Point Light oder Directional Light geladen,

- **Controls:**

- Der Charakter wird mit den WASD Tasten und der Maus bedient. Mit der Leertaste oder E kann man springen. Interaktionen, wie zum Beispiel das Aufheben eines Kristalls erfolgt über die F Taste. Siehe Details ganz Oben "Bewegung in Crystals"

- **Basic Gameplay:**

- Es gibt zum Zeitpunkt der Abgabe 2 Level. Das Erste Level ist auch schon für die Endabgabe gedacht, das Zweite (welches in der .exe mit Assimp geladen wird) stellt eine Art Playground dar, wo



wir in der Entwicklungsphase verschiedenste Funktionen testen können.

○

- **Adjustable Parameters:**

- Parameter werden mit einem settings.ini und für jedes Level und damit jedes Objekt mittels einem X\_level.ini (X..Levelnummer) geladen (Betrifft auch shader Einstellungen).

- **HUD**

- Das HUD wurde mittels FreeType implementiert. Dabei wurden die ersten 128 ASCII Zeichen geladen und verwendet sowie neue vertex und fragment shader geschrieben. Momentan gibt es noch einen Transparenz Bug, wodurch die gerenderten quads eine andere Hintergrundfarbe haben als der eigentliche Hintergrund der Szene.

- **Game-Logik**

- Die Game Logik besteht aus einer Klasse GameObjData als gemeinsame Oberklasse der 9 spezielleren Unterklassen. GameObjData enthält Informationen welche für jedes importierte Objekt nützlich sein kann (z.B isReadyToDraw, oder hasBakedTexture uvm.)
- Die Unterklassen (kurz umrissen):
  - Door (GameObjDoor):
    - ist ein Tür Objekt, welches gelocked sein kann und man mit gewissen Umständen auch unlocked um in das nächste Level zu gelangen.
  - Static: Statische Objekte ohne wesentlicher Funktion
  - Spawn: Für Spawn Punkte
  - Functional Crystal: (in Level 2 z.B)
    - Kann für Functional Logic Unit Gates verwendet werden.
    - Enthält Eigenschaften eines func. Crystal Objektes
    - Wie oft verwendbar, wie oft benutzt, usw.
  - Key Crystal:
    - Kann für Key Logic Unit Gates verwendet werden (Key-LUT)
    - Einfacher als func. Crystal. Enthält jedenfalls Informationen darüber ob dieser schon einmal benutzt wurde etc.
  - Health Crystal:



- Nicht für LUTs verwendbar. Bringt dem Character beim aufheben ein weiteres Leben.
- LUTs (Logic Unit Gates)
  - Enthält Informationen über einzelne LUTs, welche Crystals verwendet werden dürfen, welche Kinematics oder Türen dadurch getriggert werden usw.
- Kinematic
  - Enthält Informationen über die angezielte Animation des Objektes, welche im Falle einer Auslösung (oder immer wenn so angegeben) gestartet wird. Enthält auch Informationen darüber ob es ausgelöst wurde und ob es sich gerade bewegt usw.
- Character
  - Enthält alle Character typischen Informationen, wie z.B.: wie viele Leben, Crystals hat dieser etc.
- Mit diesen Eigenschaften ist es im Spiel möglich, dass Objekte untereinander kommunizieren oder Pointer zueinander abspeichern uvm. können.

## Effekte

### ● GPU-Particles

- Bei den Partikeln wurden die Folien aus dem Repetitorium verwendet. Dazu wurde eine neue Klasse angelegt, "GPUparticles". Im Konstruktor werden 2x2 SSBOs angelegt, jeweils für den In- und Output der Positions und Geschwindigkeits Buffer. Weiters wurde ein atomic counter erstellt und einige anfängliche Parameter hinzugefügt. Anschließend wurde ein neues VAO erstellt und abschließend die Textur gebündelt.

Die beiden Methoden computeShaderM und particleShaderM erstellen die 4 notwendigen Shader (vertex, fragment, geometry und compute) und fügen sie zum Programm hinzu. Die 4 Shader wurden nach den Folien Angaben geschrieben.

Die calculate Methode berechnet anfangs das dynamische Erstellen von den Partikeln und setzt anschließend einige Uniforms. Anschließend werden die SSBOs und der atomic counter gesetzt und der compute Shader mit einer vordefinierten Arbeitsgruppengröße, in unserem Fall 100x100, ausgeführt. Die Ergebnisse werden abschließend aus den Buffern rausgelesen.



Die draw Methode setzt abermals einige Uniforms, welche vor allem für die Positions der Partikel wichtig sind und führt das Programm particleShader aus, welches die Partikel zeichnet.

Allerdings funktionieren die Partikel leider nicht, es passiert nämlich nichts. Für Testzwecke wurde der Partikel Effekt auf die Taste P gebündelt, jedoch tut sich nichts im Spiel außer einer Fehlermeldung, welche aber nicht zum Spielabsturz führt. Um diesen Effekt auszuprobieren und die Fehlermeldung zu sehen müssen in der main loop die Kommentare der Partikel Zeilen entfernt werden sowie weiter oben außerhalb der Schleife bei der Definition der shader die Kommentare bei particleShader entfernt werden. Daraufhin sieht man im Spiel beim betätigen der P Taste die entsprechende Fehlermeldungen, welche sich auf den atomic counter bezieht.

- **Cel-shading + contours (backface)**
  - Für das Cel-Shading wurden die bereits im Framework vorhandenen vertex und fragment shader angepasst, indem eine neue Konstante "levels" eingebaut wurde, welche die Anzahl der verschiedenen Helligkeitsstufen darstellt. Weiters wurde die phong Methode erweitert, indem der Helligkeitswert von den diffuse und specular Koeffizienten mit der Anzahl der Level multipliziert wird, dieser Wert dann anschließend abgerundet und abschließend durch die Anzahl der Level dividiert wird. Um das cel shading zu aktivieren muss man in den settings unter shading von NORMAL auf CEL stellen.
- **Procedural Textures**
  - Procedural Textures kommen im 3ten und letzten Level zum Einsatz. Mit einem Brick Wall shader werden prozedurale Ziegelwandtexturen auf Wände gerechnet (Nur im Normal Shader Mode).
- **Watermesh**
  - Das Watermesh wird mit der Game-Logik schon beim importieren über das level.ini File als solches erkannt. Damit bekommt es ein eigenes Material und es werden zusätzliche Berechnungen im Vertex shader dafür ausgeführt.

## Tools

Zur Erstellung der Modelle wurde Blender verwendet.  
Texturen wurden teilweise mit Photoshop bearbeitet



## Features

- Funktionale PhysX
- PhysX Meshcooking
- komplexes Modelloading mittels assimp
- HUD
- Ausgefeilteres Game - Logic System
  - Damit lassen sich ohne größere Probleme immer wieder neue Levels einfach importieren
- Light Maps
- View-Frustum Culling
- Water Mesh

## Inkludierte Bibliotheken

- Glew -> <https://github.com/nigels-com/glew>
- Assimp -> <https://github.com/assimp/assimp/archive/v4.1.0.zip>
- PhysX -> <https://github.com/NVIDIAGameWorks/PhysX-3.4>
- ECG\_Library
- FreeType 2.0 -> <https://www.freetype.org/download.html>

