Alexandra Gamsjäger, Susanne Rinortner

Get Lost

Bibliotheken:

Blender und Assimp:
Um die Szene unseres Spiels einfach zu implementieren, haben wir uns dafür entschieden diese mittels Blender zu modellieren [1] und über die Assimp-Library in unser Programm zu laden. [2]
Blender Version: 2.79
Assimp Version: 4.1.0

Hier werden die notwendigen Werte der Blender-Datei, wie Positionen, Normalen, UV-Koordinaten, Texturen und Materialien geladen und in ein GameObject gespeichert, welches dann auch in der Render-Loop über eine draw-Methode aufgerufen und gezeichnet wird, wobei hier jedes Mesh für sich gerendert wird.

FFMPEG: [3]
Wurde für die Video-texturen verwendet.
Version: 4.0

FreeType: [4]
Wurde für das HUD verwendet.
Version: 2.9.1

NVIDIA PhysX:[5]
Wurde für Collision-Detection und Schwerkraft verwendet.
Version: 3.4

Implementierungen:

- **HUD**: Implementierung mittels Freetype.
- **Light Mapping**: Hier wurde das Licht direkt in Blender in die Textur gebaked.
- **Physics Engine:** Jedes Objekt hat einen statischen oder einen dynamischen Actor. Die Kamera enthält einen Character Controller, der mit den Actors kollidiert.
- **Complex 3D Models from Files:** Objekte werden mit Assimp im Collada Format geladen und enthalten dabei schon eine UV Map sowie Normalen und Texturen.
- **Animation Using Physics Engine:** Die Hitboxen der Magic Tokens bewegen sich mit der Schwebe-Animation der gerenderten Tokens mit. Diese können mittels Collision Detection aufgesammelt werden.

---

[1] https://www.blender.org/ am 30.4.2018
[2] http://www.assimp.org/ am 30.4.2018
[3] https://ffmpeg.zeranoe.com/builds/
[4] https://github.com/ubawurinna/freetype-windows-binaries
[5] https://developer.nvidia.com/physx-source-github

- **Frustum Culling:** Mittels Clipping Planes realisiert. Alles außerhalb der Planes wird nicht gerendert.
- **Controls:** Die Position der Kamera/des Charakters wird über die Tasten W(nach vorne)A(nach links)S(nach hinten)D(nach rechts) gesteuert, diese Tasteneingaben werden mittels Polling über jede Iteration der Render-Loop abgefragt und an die Kameraklasse übergeben (update-Methode), um dann die Kameraposition passend verändert. Um auch die Rotation des „Charakterkopfes“ zu simulieren, um die Welt nach belieben betrachten zu können, wird auch hier mittels Polling die Cursorposition in der Render-Loop abgefragt und dem Kameraupdate übergeben. Hier wird dann die Rotation um die X- und Y-Achse (Pitch, Yaw) angepasst und die Kamerarichtung (direction) verändert.Wird nun mittels „W“ gelenkt bewegt sich der Charakter/die Kamera in Blickrichtung, welche über die Maus/den Cursor eingestellt wird. Das selbe gilt für den Rückwärtsgang mittels „S“ in die gegengesetzte Blickrichtung und bei einer Bewegung zu einer Seite wird auch über die Blickrichtung bzw. die rechte und linke Seite dieser die Position der Kamera verändert. Des Weiteren ist es möglich über die Leertaste den Charakter/die Kamera springen zu lassen. Wobei ein anhaltender Druck auf diese ein auf und ab springen und ein einzelner Tastendruck auf die Leertaste einen einmaligen Sprung initiiert.
- **Debug Options:**
    - F1 - Hilfe
    - F2: Frames per Second
    - F3: Wireframe
    - F4: Backface-Contouring on/off
    - F8: View-Frustum Culling on/off

- **Gameplay:**Das Spiel beginnt am Eingang des Labyrinths, von hier kann man sich durch den Irrgarten bewegen bis man zu einer braunen Tür am anderen Ende kommt. Beim Durchlaufen des Labyrinths kann Man Tokens einsammeln die einem Magische Kraft bringen. Zu dem Muss man Hindernisse in Form von Löchern im Boden überspringen. Durchschreitet man die Tür kommt man in einen weiter unten platzierten Raum in welchem die Prinzessin auf einen wartet. Hier wird man von Flammenwerfern begrüßt die man entweder mit dem Verlust Magischer Kraft durchschreiten oder sich seitlich auf der Mauer zur Prinzessin vorbeischleichen kann. Die Prinzessin die nicht gerettet werden will ist somit der Endboss. Berührt man sie mit einem geringeren Magiewert als 15 ist man sofort tot. Nur wenn man sie mit einem Magiewert der Größe 15 berührt kann man gewinnen.

Features:

Am Weg durchs Labyrinth kommt man bereits an Löchern im Boden vorbei welche übersprungen werden müssen. Fällt man in ein Loch werden die Leben auf -1 gesetzt und der Spieler verliert.
Ebenso sind bereits die schwebenden Magic-Tokens eingesammelt werden können. Es existiert auch eine Plattform die mittels Sprung erreicht werden kann.
Auf der Plattform befindet sich ein extrem magischer Token der mehr magische Kraft bringt.

Es gibt auch Hindernisse in Form von Flüssen welche nicht überwunden werden können und den Spieler zwingen einen anderen Weg durch das Labyrinth zu finden.
Hat man den Weg durch das Labyrinth gefunden sieht man Flammenwerfer (Partikelsystem) die die restliche Welt vor der Prinzessin schützen.

Belichtung:

Auch Materialien werden in die Blender-Datei eingebunden und mittels Assimp-Library in die verschiedenen GameObjectMeshes gespeichert.
Da es sich bei unserer Lichtquelle, um eine Sonne handelt, haben wir uns für ein einfaches DirectionalLight entschieden, wobei dieses über der Szene schwebt und gleichmäßig Licht auf das Labyrinth fallen lässt.
Des weiteren enthält die Szene ein Point Light welches das Zentrum des Labyrinths stärker ausleuchtet.
Für die Wände des Endboss-Rooms wurden Lightmaps verwendet da diese statisch sind und sich die Beleuchtung nicht verändert.
Da auf die Wände des Labyrinths Simple Normal Mapping angewendet wird haben wir uns dort gegen Lightmaps entschieden.

Effekte:

- **Simple Normal Mapping:**[6] Über die Blender-Dateien werden schon im voraus die Normal-Maps den einzelnen Meshes zugewiesen und im Programm mittels Assimp geladen. Die gegebenen Normal-Maps werden dann als Textur in den Shader geladen und zu den original Normalen des Objekts hinzugezählt um den Effekt des Simple Normal Mappings zu erzeugen. Der zu hinzufügende Vektor wird über die RGB Daten der Normal-Map-Textur übergeben. Dadurch scheinen die Texturen im Spiel realisischer. Zurzeit auf den Wänden des Labyrinths implementiert.
- **Backface - Contouring:** Mittels der schon gegebenen Funktionen glCullFace(GL_BACK) und glCullFace(GL_FRONT) werden vorerst die Backfaces über einen einfachen PhongShader schwarz sowie größer gezeichnet, daraufhin werden die Frontfaces über einen TextureShader in original Größe gezeichnet. Dieser Effekt wurde aus ästhetischen Gründen nur für die Wände implementiert.
- **Video Textures:** [7] Werden mit der Bibliothek ffmpeg geladen, decodet und in einzelne Frames geteilt.Jedes Frame wird ein neues VideoFrame auf die MagicTokens gerendert. Für die VideoTexturen wurde eine eigene Klasse erstellt um das ändern der Textur auf den Tokens zu ermöglichen.
- **Water Mesh:**[8] Hierfür wurden Planes erstellt. Diese wurden dann in Blender sub-devided um mehr Vertices zu bekommen. Anschließend wurden diese Vertices im Vertex-Shader in einer Sinuskurve verschoben und die Normalen mittels der

[6] https://learnopengl.com/Advanced-Lighting/Normal-Mapping
[7] http://dranger.com/ffmpeg/
[8] https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/Animation_SS18.pdf

Ableitung der Sinus Funktion neu berechnet. Durch Blending und des Ändern des Alpha werts wurde ein Durchscheinender Effekt erzielt.

- **Particle System mit Compute Shader und Instancing:**[9] Für die Particles werden Punkte mittels Geometry shader in Quads verwandelt welche im Fragment shader zu glühenden Bällen gerendert werden. Zum Updaten wurde ein Compute shader verwendet und gerendert wird mit Instancing.

[9] https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/ComputeShader_SS18.pdf, https://tuwel.tuwien.ac.at/pluginfile.php/1025327/mod_page/content/26/Particles_SS17.pdf, https://www.cg.tuwien.ac.at/courses/Realtime/repetitorium/index.html,