

Dokumentation „Vlad fladert“

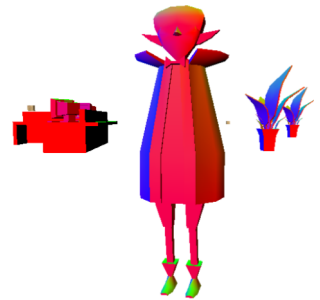
Implementierung:

EFFEKTE:

Cel Shading (0.5 Punkte): Diffuse und Specular Teile werden nicht smooth geschadet, sondern in sichtbaren Level Abstufungen dargestellt. Im Fragment Shader wird dafür für jedes vorhandene Licht einzeln der Helligkeitswert auf eines der Levels abgebildet. Die Anzahl der Levels sind fix, und können im Shader angepasst werden. Da mehrere Lichter, die auf ein und dasselbe Objekt treffen oft unschöne Ergebnisse liefern, vor allem um so mehr Levels verwendet werden, sind sowohl Lichtanzahl als auch Levelanzahl niedrig gehalten worden.

(Effekt sichtbar auf allen Wachen und Vlad)

Outlines als Post Processing Effekt (1 Punkt): Um Outlines zu berechnen habe ich einige gute Ansätze gefunden. So kann ein Kantenfilter zum Beispiel auf Normalen oder Depthbuffer Werte, oder eine Kombination aus beidem angewandt werden. Ich habe mich für die Variante eines Sobel Filters auf die Normaleninformation entschieden. Dafür werden in einem Renderpass die Richtungen der Normalen der einzelnen Fragmente als Farbinformation in einem Framebuffer abgespeichert, und in einem zweiten Schritt darauf ein Sobelfilter angewandt. An Stellen, an denen eine abrupte Änderung der Normalen festgestellt werden kann, wird somit eine Kante eingezeichnet. Der einzige Nachteil dieser Methode: sind zwei Objekte mit gleicher Normalenrichtung genau hintereinander, werden keine Outlines gezeichnet.



[1][2][3]

CPU Particles mit Instancing (0.5 Punkte):

Positionen, Größe, Bewegung über den Lauf der Zeit und Logik zum neu Erschaffen/Sterben der Partikel werden von der CPU ausgeführt, alle nötigen Informationen werden in einem Buffer gesammelt auf die GPU geladen, und dort mit einem Instanced draw Call für alle noch lebenden Partikel ausgeführt. Alle Partikel teilen sich dabei dieselben vier Vertex Positionen und UV Koordinaten. Da die Partikel einen Alphakanal haben, müssen diese nach Entfernung zur Kamera sortiert, und von „hinten nach vorne“ übereinander gezeichnet werden, da sonst unnatürliche Überblendungen durch das Alpha Blending entstehen.

Da die Partikel lediglich flache Bilder sind, müssen sie stets in Richtung Kamera gedreht werden. Dies kann mit einer einzigen ModelMatrix für alle Partikel eines Systems gelöst werden, die die Rotation der Kamera aufhebt. Alle weiteren Modifikationen der ModelMatrix (Translation und Skalierung) werden dann direkt im Vertex Shader durchgeführt.

Die Partikelsysteme werden nur gezeichnet, wenn sie im View Frustum liegen, eine Ausgabe über die Anzahl der gerenderten Partikel ist ebenfalls in der View Frustum Culling Debug Funktion vorhanden. (Effekt sichtbar bei Lebensverlust, und als Markierung des Ziels)

[4][5]

GAMEPLAY:

Das Ziel des Spiels ist es, den Wachen auszuweichen, und zum Ende des Levels zu gelangen. Gerät

man in das Sichtfeld einer Wache, wird die Szene für einen kurzen Moment „eingefroren“ - niemand bewegt sich in dieser Zeit, damit man die Gelegenheit hat, sich noch kurz umzusehen, um eventuell zuvor nicht bemerkte Gegner zu entdecken. Die Sicht Detektion ist mittels PhysX Raycast implementiert. Zur Vereinfachung der Spielmechanik werden, sofern sich Vlad in Blickrichtung der Gegner befindet (der Blick verläuft kegelförmig, mit einem Öffnungswinkel von 90 Grad) zwei Raycasts, die parallel zum Boden verlaufen, in Richtung Vlad geschickt. Einer davon in Kopfhöhe, und einer auf Kniehöhe. Diese Variante bietet den Vorteil, dass einfacher abschätzbar ist, ob Vlad gerade sichtbar ist, oder nicht, auch wenn sie dadurch weniger realistisch ist.

Die Raycasts vermerken nur den ersten Kontakt mit einem PhysX Actor. Ist dies der Actor „Vlad“ - in der PhysX Simulation als kapselförmiger Character Controller dargestellt – verliert dieser ein Leben und muss zurück zum Start. Der Character Controller wird beim Ducken auf die halbe Höhe skaliert.

Versteckmöglichkeiten und Architektur des Levels wurden sehr vereinfacht in Blender modelliert, und die Triangle Meshes für die PhysX Simulation gecooked. In der Renderengine werden sie dann allerdings durch detailreichere Möbel u.s.w ersetzt.

[6][7][8]

WEITERE ERFORDERLICHE FEATURES:

Complex Models from Files: Die Figuren und Möbel sind alle in Blender erstellt, texturiert und exportiert worden, und werden mit einem selbst implementierten .obj Loader beim Startup eingelesen. Alle Modelle und Texturen wurden selbst gemacht.

Light Mapping: Light Mapping wurde für alle statischen Objekte in Blender durchgeführt (aufgrund der niedrigen Auflösung der Lichtinformationen habe ich vor, die Texturen bis zum Spielevent nochmals mit höherer Sample Rate zu baken.)

Hierarchical Animation Using Physics Engine: Der Hauptcharacter bekommt seine Koordinaten durch die Simulation der Nvidia Physx Engine. Somit kann er mit Möbeln und Wänden kollidieren und Rampen hinaufgehen. Er besteht aus insgesamt sechs Teilen (Körper, Kopf, 2x Oberschenkel und 2x Unterschenkel), die Modelmatrizen vor den draw Calls aneinander weitergeben.

Die gleiche Logik wird auch bei den patrouillierenden Wächtern eingesetzt, nur dass diese ohne Einbindung in die Physx Engine funktionieren, da bei ihnen der Einfachheit halber keine Steigungen oder Kollisionen mit Objekten vorgesehen sind. Lediglich der Raycast Test, der feststellt, ob der Spieler entdeckt wurde, wird von ihrer Position aus entsandt, und mithilfe von Physx umgesetzt.

View Frustum Culling: Für das View Frustum Culling wurde jedes Spielelement mit einem berechneten Radius und Mittelpunkt versehen. Die durch eine Kugel approximierte Form wird dann nach Verschiebung durch die ModelMatrix auf Lage zu den sechs Planes, die das Kamera View Frustum ausmachen, geprüft. Die sechs Planes müssen nach jeder Bewegung der Kamera neu berechnet werden. [9]

Hinweis: Da Vlad und das Krankenhaus in jedem Frame zu sehen sind, und immer gezeichnet werden müssen, sind sie aus dem Counter ausgenommen.

Debug Options: Alle Debug Options mit Ausgabe (Frustum Culling, Frametime) werden mithilfe der Library Freetype dargestellt.

[10]

Heads Up Display: Das HUD besteht aus der Anzeige der verbleibenden Leben, gerendert auf einzelne Quads, und eventuellen WIN/LOSE Status Nachrichten, die wie die Debug Optionen mithilfe von Freetype ausgegeben werden.

Controls: „Vlad“ ist mit der Tastatur steuerbar:

W	Vorwärts bewegen
A, D	Nach links/rechts rotieren
S	Ducken/Aufstehen – im Moment nur provisorisch vorhanden, steuert Geschwindigkeit, und Rotation des Platzhaltermodells
Leertaste (halten)	Sprinten – nur möglich, wenn Vlad sich nicht gerade duckt

Zusatztasten:

F	Toggelt Fullscreen/Windowed mode
G	Toggelt „Godmode“ - der Spieler kann zwar noch von den Gegnern detektiert werden, Teleports zurück zum Start, und Lebensabzüge werden aber nicht ausgeführt.
Pfeiltaste hoch bzw. nach unten	Gesamthelligkeit erhöhen/vermindern
F1	Hilfe ein/aus – zeigt die Steuerung an
F2	Framerate ein/ausblenden
F3	Wireframe ein/aus
F8	Toggelt Counter für Elemente im View Frustum, die tatsächlich gezeichnet werden.

Adjustable Parameters:

Screen Resolution: Anfangswert wird über das settings.ini file (assets/settings.ini) eingelesen, das Fenster kann vom User während dem Spiel vergrößert/verkleinert werden, jedoch ist der anfängliche Aspect Ratio unveränderlich.

Fullscreen: Die Taste „F“ toggelt Fullscreen/Windowed Mode

RefreshRate: wird aus dem settings.ini file ausgelesen.

Brightness: Ein Uniform, das sich in allen Shadern befindet kann mit Drücken der Pfeiltasten nach oben oder unten zusätzliche Helligkeit hinzufügen und wieder entfernen.

Shader: testshader.vert/testshader.frag

Celshader, nur für Wächter und Vlad verwendet

Lichter, die vom Shader verwaltet werden:

ein Directional Light und 5 Spotlights (90 Grad outer Angle, 80 Grad inner Angle)

Objekte, die damit beleuchtet werden:

Vlad

und die 3 Guards

Shader: bakedTextures.vert/bakedTextures.frag

Lichter:

Objekte, die damit gerendert werden:

Alle Pflanzen/Möbel/Wände/Boden/Decke

Zusätzliche Features: .obj Loader – selbst implementierte Version, liest Vertex Positionen, Normalen, UV Koordinaten und Indizes aus dem File und speichert sie in den entsprechenden VBOs. Berechnet zusätzlich Radius und Mittelpunkt des Objekts.

Quellen:

- [1] DECAUDIN, Philippe. Cartoon-looking rendering of 3D-scenes. Syntim Project Inria, 1996, 6. Jg.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.6652&rep=rep1&type=pdf>
- [2]<http://research.cs.wisc.edu/graphics/Courses/AdvancedGraphics09/Yangk/P1Extra>
- [3] <https://learnopengl.com/Advanced-OpenGL/Framebuffers>
- [4] Folien über Partikelsysteme aus dem SS17 (TUWEL)
- [5] <https://learnopengl.com/Advanced-OpenGL/Instancing>
- [6]<https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/SceneQueries.html>
- [7]<https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/CharacterControllers.html>
- [8]<https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Geometry.html>
- [9]<http://www.lighthouse3d.com/tutorials/view-frustum-culling/>
- [10] <https://learnopengl.com/In-Practice/Text-Rendering>