Optimierung von Voxel Engines mittels Culling und Greedy Meshing

MINT Hausarbeit an der Martin Luther Schule

Arne Daude

24. Oktober 2024

Inhaltsverzeichnis

1 Einleitung

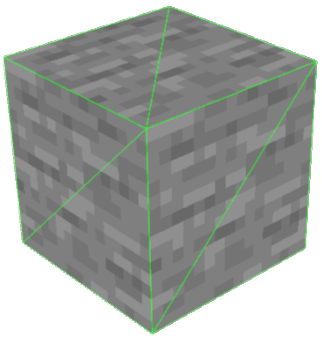
Eine Voxel Engine ist zuständig Voxels auf den Bildschirm zu projizieren. Voxels sind dabei wie Pixel, nur 3D (der Name „Voxel“ kommt von „Volumen“ kombiniert mit „Pixel“). Also werden viele kleine Würfel zusammengesetzt, um ein drei dimensionale Objekt darzustellen.

Dies wird in manchen Videospielen benutzt, wie Minecraft. Als Spieleentwickler erstelle ich auch ein Voxel Spiel und bin auf das Problem getreten, wie man diese implementiert. Es gibt hauptsächlich zwei Varianten, wie Voxel Engines implementiert werden:

- Volumengrafik: Es wird ein komplett neues Verfahren entwickelt, um die Voxels zu rendern, welches direkt mit den Voxels arbeitet. Dies hat zwar gute Performance, hat aber auch die Limitation, dass alle Voxel nur einfache Würfel seien können.
- Polygonnetz: Die Voxels werden zuerst in viele Dreiecke (oder allgemein Polygone) umgewandelt und dann von einem typischen 3D Renderer angezeigt. Somit muss man keinen neuen Renderer erstellen und kann auch andere Modelle als nur Würfel anzeigen, aber dieses Verfahren hat dafür schlechtere Performance.

Wegen der einfacheren Implementation, und dass man komplexere Modelle als nur Würfel erstellen kann, werden in den meisten Spielen die Polygonnetz Variante bevorzugt.

Mit der Polygonnetz Implementation sieht dann ein Voxel wie rechts gezeigt aus. Die grünen Linien zeigen dabei wie es in Dreiecke eingeteilt ist.



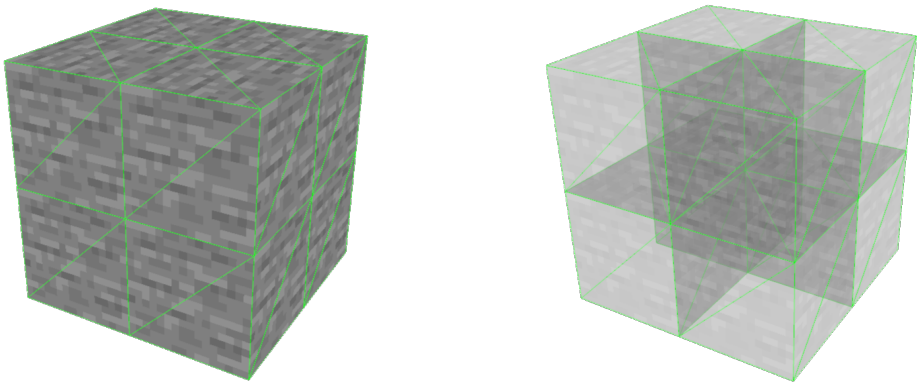
Ein Quadrat wird aus 2 Dreiecken zusammengesetzt und ein Würfel hat 6 Seiten. Somit besteht ein Würfel aus 12 Dreiecken. Wenn der Spieler nur 100 Voxels weit sehen könnte, wäre der Durchmesser 200 Voxels und somit das gesamte Volumen das angezeigt werden muss $200^3 = 8.000.000$ Voxels groß, also $12 \cdot 200^3 = 96.000.000$ Dreiecke.

Wir sehen also, dass schon bei einer sehr kleinen Sichtweite sehr viele Dreiecke erstellt werden. Diese Hausarbeit beschäftigt sich deswegen mit der Optimierung die Anzahl der Dreiecke so weit wie möglich zu reduzieren.

Die meisten Optimierungen in dieser Hausarbeit kommen von https://youtu.be/qnGoGq7DWMc?si=Ke8vgcHWcgCdMGka und https://github.com/TanTanDev/binary_greedy_mesher_demo, und wurden dann noch ausgeweitet, damit sie auch für Voxels mit Texturen funktionieren.

2 Culling

Der erste Weg eine Voxel Engine zu optimieren wird offensichtlich, wenn man sich einen Würfel aus 8 Voxels betrachtet. Dabei beobachten wir, dass die Hälfte der Seiten der Voxels nach innen schauen und somit nicht sichtbar sind. Wenn man die Seitenlänge dieses Würfels verdoppelt, multipliziert man die Oberfläche mit 4 und das Volumen mit 8. Somit entstehen immer mehr Seiten, die nach innen schauen, umso größer das Objekt ist. Also wird dies bei großen Objekten dazu führen, dass die meisten Seiten nicht sichtbar sind.



Mit „Culling“ beschreibt man die Optimierung, diese Seiten zu entfernen.

3 Greedy Meshing

4 Fazit

5 Quellen / Literaturverzeichnis

- https://youtu.be/qnGoGq7DWMc?si=Ke8vgcHWcgCdMGka
- https://github.com/TanTanDev/binary_greedy_mesher_demo
- https://de.wikipedia.org/wiki/Volumengrafik
- https://de.wikipedia.org/wiki/Polygonnetz

Der Quellcode für dieses LaTeX Dokument:
https://github.com/BlueSheep3/MINT-Hausarbeit.

Meine Implementation dieser Optimierungen:
https://github.com/BlueSheep3/voxel_game.