

Spieleprogrammierung - Übung 1

Marlin Zapp

25. Mai 2025

1 Einleitung

Im Laufe dieser Übung habe ich deutlich gemerkt, dass die Wahl der passenden Engine nicht ganz leicht ist. Um eine Entscheidung zu treffen, habe ich ChatGPT gefragt, welche Engines Raytracing unterstützen. Die KI hat mich auf Unity, Unreal und Godot verwiesen, mit dem Hinweis, dass Godot Raytracing nur experimentell unterstützt. Da Godot freie Software und somit auf Linux leicht installierbar ist, habe ich mich zuerst kurzerhand für diese Engine entschieden.

An dieser Stelle ist folgende Diskussion zwischen ChatGPT und mir recht interessant.

Is there an option for ray tracing the lighting in godot? And if so, how do i use it?

Yes, Godot does have options for ray-traced lighting, but it depends on the version you're using. Godot 4 introduced a full Vulkan renderer, and ray-traced lighting support is included — but it's still considered experimental.

It's called Ray-Traced Global Illumination (RTGI) and Ray-Traced Reflections, among other features.

Enable ray tracing in Godot 4 in the Project Settings under *Rendering → Features → RayTracing*.

Beim Ausprobieren dieser Anweisung war schnell klar, dass die Antwort nicht ganz stimmt. Es gibt keine Einstellung *Rendering → Features → RayTracing* in Godot 4. Nach kurzer Recherche habe ich außerdem herausgefunden, dass Godot kein Feature namens Ray-Traced Global Illumination oder Ray-Traced Reflections besitzt. Infolgedessen habe ich einige Zeit investiert, um Diskussionen rund um die Implementierung von Ray-Tracing in Godot zu verfolgen.

Momentan ist es so, dass Godot verschiedene Methoden anbietet, um möglichst realistische Beleuchtung zu erhalten. Keine davon ist ein Echtzeit-Ray-Tracing. Seit Vulkan Ende 2020 ein Software Development Kit veröffentlicht hat, gibt es in der Godot-Community den Vorschlag, Vulkan Ray-Tracing zu implementieren. Es gibt zwar schon einen Pull-Request, aber leider ist die Implementierung noch weit davon entfernt, in einem Renderer zu landen.

Obwohl die Engine kein echtes Ray-Tracing anbietet, habe ich mich dazu entschieden, die diversen Beleuchtungstechniken, die Godot zur Verfügung stellt, auszuprobieren. Die Techniken verfolgen einige interessante Ansätze, die ich in Abschnitt 3 anhand meiner Tests vorstellen möchte.

Vorher gehe ich jedoch in Abschnitt 2 auf meinen Code zur Labyrinth-Erstellung und dessen Interaktion mit der Engine ein.

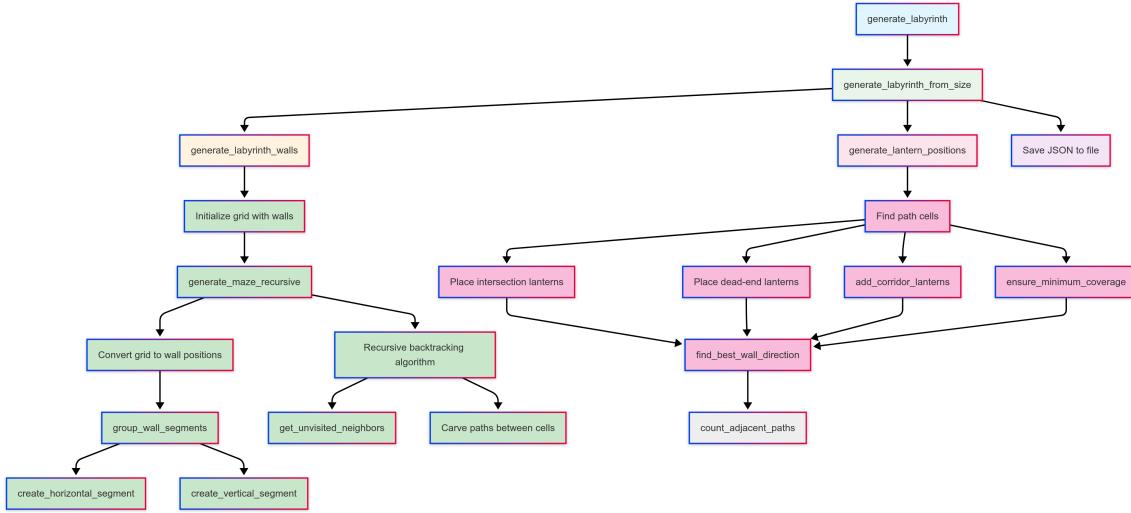


Abbildung 1: Flowchart zum Ablauf der Labyrinth-Generierung

2 Labyrinth

Bezüglich des Programmierens ist die Labyrinth-Generierung der interessanteste Teil dieser Übung. Ich habe mich dazu entschieden, das Labyrinth in Form einer JSON-Datei festzuhalten. Das hat zwar den Nachteil, dass man bei einem kurzen Blick auf die Datei schlecht erkennt, wie das Labyrinth aussieht, aber unter anderem den Vorteil, dass man größere Wand-Segmente anhand von Start- und Endposition definieren kann.

Die Generierung der JSON-Datei erfolgt mithilfe des Scripts „generate_labyrinth.gd“ in der Godot-eigenen Programmiersprache GDScript. Dieses Script ist an die Root-Node der Szene angehängt und kann im Godot-Editor ausgeführt werden (siehe Abbildung 2). Eine Übersicht des Ablaufs der Generierung findet sich in Abbildung 1. Ich habe zuerst versucht, das Script mithilfe von ChatGPT zu schreiben, aber habe es nicht geschafft, eine sinnvolle Ausgabe zu erhalten. Deshalb bin ich dafür das erste Mal auf Claude Sonnet umgestiegen und habe mithilfe dieser KI tatsächlich deutlich bessere Ergebnisse erzielt.

Wenn man die neue oder eine andere JSON-Datei nutzen möchte, kann man sie mit dem Script „load_labyrinth.gd“ einlesen und das Labyrinth im Editor erzeugen. Der Button dafür befindet sich auf der Node „LabyrinthWalls“. Der große Vorteil daran, das Labyrinth nicht zur Laufzeit, sondern mit einem Editor-Skript zu erzeugen, ist, dass das Labyrinth schon im Editor einzusehen ist. Dadurch lassen sich z.B. statische Lichteffekte berechnen.

3 Beleuchtungs-Techniken

Ich habe die vier grundlegend unterschiedlichen Beleuchtungstechniken, die Godot zur Verfügung stellt, anhand von zwei Ausschnitten aus einem Labyrinth verglichen (siehe Abbildung 3). Für bessere Ergebnisse habe ich als Ergänzung zu jeder Beleuchtungstechnik SSIL (Screen Space Indirect Lighting), SSR (Screen Space Reflections) und SSAO (Screen Space Ambient Occlusion) aktiviert.

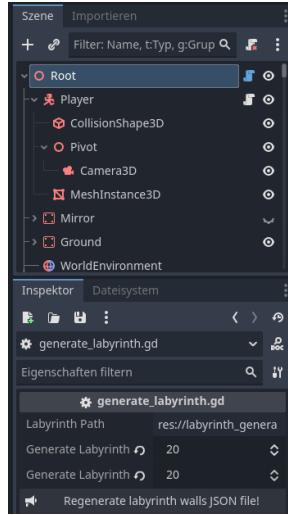


Abbildung 2: Im der Szene muss die Root-Node ausgewählt sein, damit man im Inspektor die Labyrinth-Generierung starten kann.

Lightmap GI: Ist für Low-End-Geräte geeignet und wird einmal im Voraus berechnet. Die indirekte Beleuchtung ist ganz gut, aber es gibt quasi keine Reflektionen.

Voxel GI: Wird in Echtzeit berechnet, aber funktioniert nicht sehr gut für das Labyrinth. Das liegt daran, dass für diese Technik eine VoxelGI-Node an den Raum angepasst werden muss, was für Labyrinthe fast unmöglich ist. Deshalb sind die Reflektionen im rechten Ausschnitt von Abbildung 3b zwar schön, aber einfach falsch.

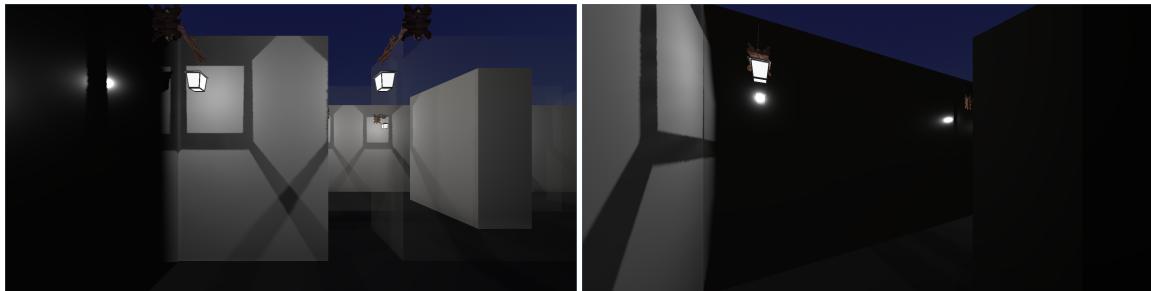
Reflection Probe: Diese Technik legt besonders Wert auf Reflektionen, aber ist noch anfälliger für die Labyrinth-Struktur als VoxelGI. In den beiden Bildern sieht das Ganze zwar gut aus, aber wenn man mit dem Charakter durch das Labyrinth läuft, dann sieht man manchmal Reflektionen von Lichtern, die eigentlich hinter Wänden verborgen sind. Dies lässt sich nur beheben, wenn man für jeden Gang eine eigene Reflection Probe Node anlegt.

SDFGI (Signed distance field global illumination): Diese Technik ist am einfachsten einzusetzen und führt in dem Labyrinth zu den besten visuellen Ergebnissen. Die Reflektionen sind zwar nicht stechend scharf, aber trotzdem gut als solche erkennbar. SDFGI ist vergleichsweise anspruchsvoll für die Graphikkarte.

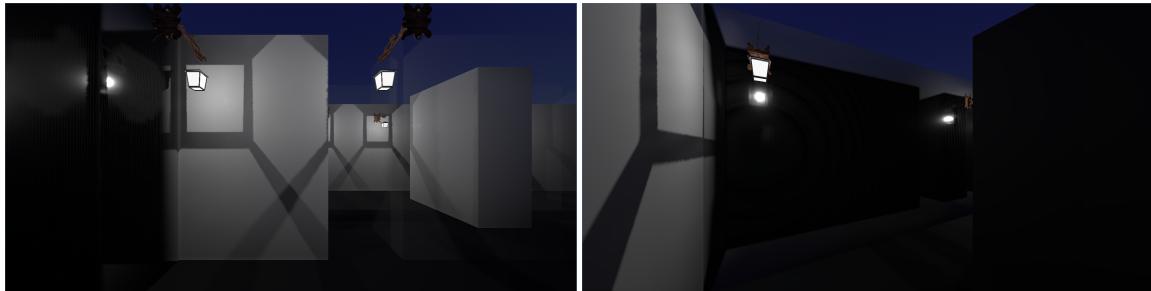
4 Fazit

Obwohl Godot bislang kein echtes Ray-Tracing ermöglicht, sind trotzdem beeindruckt gute visuelle Resultate von direkter und indirekter Beleuchtung sowie meist ausreichend gute Resultate bei Reflektionen möglich. Besonders mit der Methode SDFGI erreicht man ohne viel Konfiguration sehr gute Ergebnisse. In der Handhabung finde ich Godot meist intuitiv und das Script-Schreiben war mithilfe von Claude Sonnet sehr angenehm.

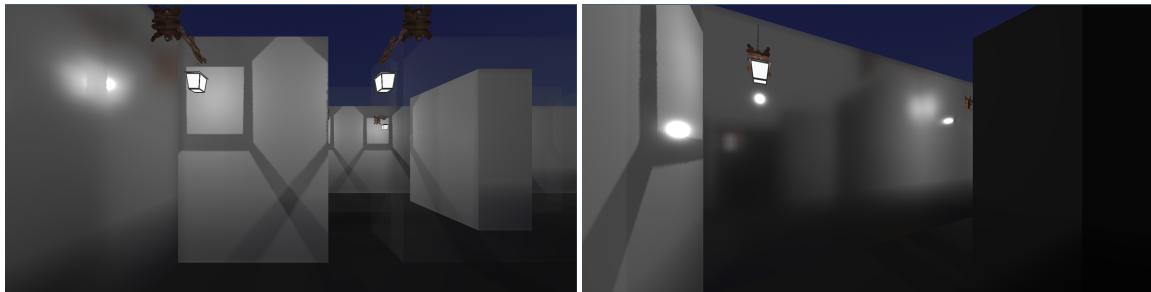
Falls jedoch das letzte Bisschen aus der Beleuchtung herausgeholt werden soll, dann muss man auf Unity oder Unreal Engine umsteigen und das dort vorhandene Hardware-Ray-Tracing nutzen.



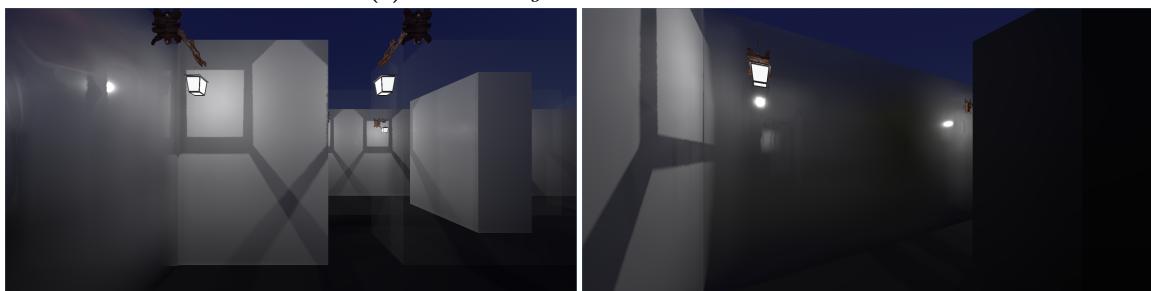
(a) Beleuchtungsart: LightmapGI.



(b) Beleuchtungsart: VoxelGI.



(c) Beleuchtungsart: Reflection Probe.



(d) Beleuchtungsart: SDFGI.

Abbildung 3: Zwei Kameraausschnitte beleuchtet durch verschiedene Techniken. *Ausschnitt links:* Metallisch reflektierende Wand links, mehrere durchsichtige Wände, schwarzer Boden, einige weiße Wände. *Ausschnitt rechts:* Weiße Wand links, metallisch reflektierende Wand geradeaus, schwarze Wand rechts, Blick durch durchsichtige Wand.