



Informatikprojekt

Konstruktion eines Renderers für implizite Oberflächen

Bastian Abt

Professor: Horst Stenzel
Betreuer: Fabian Friederichs

12. Juni 2022

Inhaltsverzeichnis

1 Einführung	3
2 Implizite Oberflächen	3
2.1 Raymarching	4
2.2 Sphere Marching	5
3 Licht und Schatten	5
3.1 Berechnung des Normalenvektors	5
3.2 Texturierung	6
3.3 Lichtberechnung	6
3.4 Schatten	7
3.5 Umgebungsverdeckung	9
4 Hierarchische Rendermethoden	10
4.1 Intervall Arithmetik	11
4.2 Parser	11
4.3 Tape	11
4.4 Renderzeiten	12
5 Audio	13
6 Fazit und Ausblick	14
7 Anhang	17

1 Einführung

Die Beschreibung von Geometrie nicht durch Daten, sondern durch Funktionen, bietet einige Vorteile, die in dieser Arbeit herausgearbeitet werden. Daher findet diese Technik Verwendung innerhalb der Bereiche Design, Modellierung, Visualisierung, Animation und in der allgemeinen Computergrafik (Raposo und Gomes 2019).

In dieser Arbeit werden einige der verwendete Techniken vorgestellt und zu einer Audiovisualisierung zusammengesetzt. Da allerdings die aktuelle Hardware auf die Darstellungen von triangulierter Geometrie optimiert ist, ist das klassische Rendern ineffizient. Keeter (2020) stellt eine Technik vor, die effizientes Rendering von impliziten Oberflächen auf aktueller Hardware erlaubt. Diese Technik wird in dieser Arbeit untersucht und die Teilkonturen Parser, Tape und Interpreter werden implementiert. Die Implementation kann unter <https://github.com/spookyGh0st/ip> abgerufen werden und wird an relevanten Stellen über Fußnoten verlinkt.

2 Implizite Oberflächen

Die Darstellung einer Szene kann nicht nur aus der Triangulation der Oberfläche bestehen. Implizite Oberflächen sind durch implizite Funktionen beschriebene Körper im euklidischen Raum. Letztere beschreiben Funktionen nicht durch explizite Zuordnung der Werte, sondern implizit durch die Gleichung $f(x, y, z) = 0$ (eingeschränkt auf 3 dimensionale Funktionen). Folglich werden Implizite Oberflächen durch die *Level Set* 0 einer Funktion im euklidischen Raum beschrieben. Die später erläuterte Technik *Sphere Tracing* erfordert eine weiter Einschränkung der Funktion zu einer Distanzfunktion, einer Isofläche, die die kleinste euklidische Distanz zu dem Körper ermittelt.

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

Im Folgenden wird von einer Distanzfunktion mit Vorzeichen ausgegangen. Diese bestimmt dabei, dass Werte > 0 Punkte außerhalb und Werte < 0 Punkte innerhalb von Objekten darstellen.

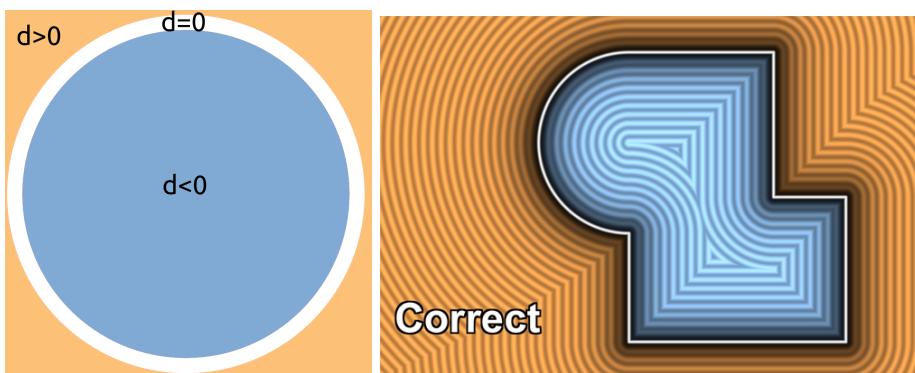


Abbildung 1: Levelmengen von Signed Distance Funktion. Die rechte Abbildung stammt von Quilez (2020). Die Linien visualisieren Level-Sets der Funktion

Da die Szene durch eine beliebig detaillierte Funktion beschrieben wird, ist es möglich eine ebenso beliebig detaillierte Geometrie zu erstellen. In dieser Arbeit wird die von (Ricci 1973) beschriebene Technik *Constructive Solid Geometry* verwendet. Einzelne geometrische Körper werden dabei durch eine Distanzfunktion definiert, wie beispielsweise eine Kugel mit einem Radius von eins durch $\sqrt{x \cdot x + y \cdot y + z \cdot z} - 1$. Komplexe Objekte werden durch die Kombination von geometrischen Körpern durch Schnittmengen und Vereinigungsmengen erstellt.

(2)

```
float cuboidD = distanceCuboid(p);
float sphereD = distanceSphere(p);
return subtraction(cuboidD, sphereD);
```

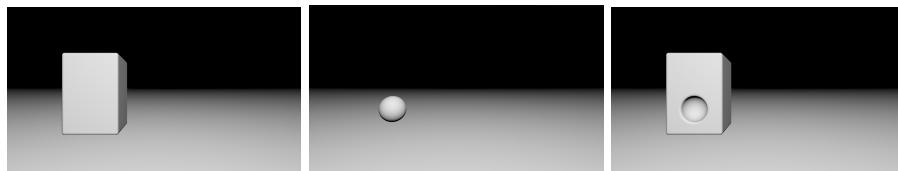


Abbildung 2: Durch Mengenoperationen lassen sich verschiedene implizite Oberflächen kombinieren. Die in diesem Projekt verwendete Szene besteht unter anderem aus der Subtraktion einer Kugel aus einem abgerundeten Quader, um eine Box darzustellen.

Die Verwendung impliziter Funktionen erlaubt eine Flexibilität, die mit Triangulation schwer zu erreichen ist. Das Kombinieren von triangulierter Geometrie durch Mengenoperationen wie in Abb. 2 abgebildet, ist äußerst komplex. Auch dynamische Änderungen von Modellen, die nicht durch lineare Transformationen beschreibbar sind, gelingen durch implizite Funktionen deutlich eleganter und leichter. Vereinfacht lassen sich auch sanfte Übergänge zwischen zwei Elementen realisieren, wie in Abb. 2 angedeutet. (Quilez 2013). Weitere mögliche Deformationen sind vielfältig und werden in dieser Arbeit nicht weiter behandelt, Sederberg und Parry (1986) geben allerdings einen guten Überblick über diese.

2.1 Raymarching

Um einen Pixel darzustellen, wird ein Strahl von dem Kameraursprung durch jeden Pixel in eine Szene hineingeschossen. Hierbei wird der erste Schnittpunkt mit der Oberfläche gesucht, um diese darzustellen. Da die Oberfläche meistens nicht meistens nicht in einer *Closed-Form Solution* gefunden werden kann, ist die Annäherung über den Strahl notwendig. Wenn man an dem Strahl entlang läuft, kann bei jedem Schritt die Distanz zur Szene berechnet werden. Sobald die Distanz einen negativen Wert erreicht hat, ist der derzeitige Punkt innerhalb eines Szenenelementes. Um die genaue Oberfläche zu berechnen, wird ein genauer Schnittpunkt zwischen den letzten beiden Punkten gefunden.

Diese Technik führt allerdings zu Problemen bei Objekten, die sehr weit oder sehr nah aneinander liegen. Bei sehr weit auseinander liegenden Objekten werden viele Schritte im leeren Raum berechnet. Bei sehr nah aneinander lie-

genden und kleinen Szenenelementen kann es leicht dazu kommen, dass diese übersprungen werden.

2.2 Sphere Marching

Ein besserer Ansatz ist das von Hart (1996) beschriebene *Sphere Marching*, bei dem jedes Mal die aktuelle Distanz zur Szene als Schrittgröße genommen wird. Hierdurch ist durch die in Abschnitt 2 besprochenen Eigenschaften auch ausgeschlossen, dass der jeweils nächste Schritt in eine Oberfläche eindringen kann. Allerdings ist die Effizienz abhängig von der entsprechenden Szene. Beispielsweise ist die Schrittanzahl entlang eines Strahls, der nahe einem flachen Element entlang zeigt sehr hoch, da die aktuelle Distanz zum nächsten Element durch das flache Element sehr gering ist. (cf. Abb. 4)¹

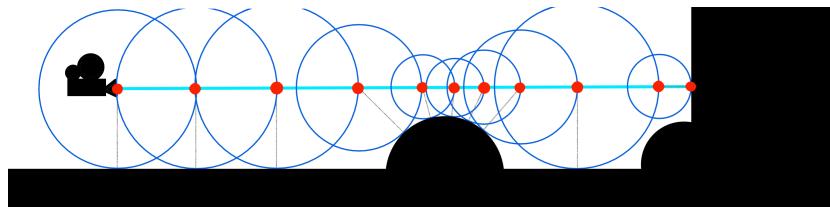


Abbildung 3: *Sphere Marching* eines einzelnen Rays

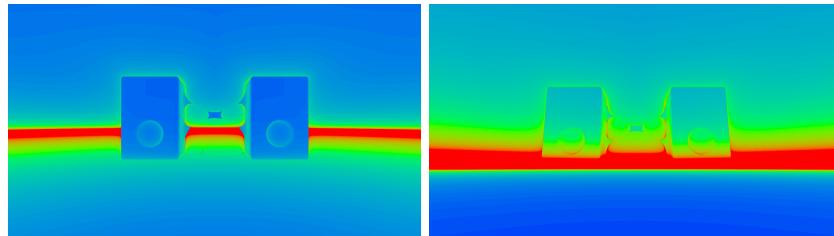


Abbildung 4: Anzahl der Schritte der Beispilszene aus unterschiedlichen Kameraspositionen. Die Farbe wird dabei die durch `stepCount` sowie `maxStepCount` bestimmt, wobei `maxStepCount = 100` gilt. Blaue Stellen haben eine geringe Schrittanzahl, rote Stellen eine hohe.

3 Licht und Schatten

3.1 Berechnung des Normalenvektors

Zur Beleuchtungsberechnung werden die Normalenvektoren der Oberflächen benötigt. Auch hier könnte eine Funktion $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ verwendet werden, um die Position auf den Normalenvektor abzubilden. In der Praxis sind solche Funktionen aber schwer bis unmöglich zu konstruieren, stattdessen wird eine elegantere Methode gewählt.

¹<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.gls1#L189>

Da der Normalenvektor orthogonal von der Oberfläche absteht, werden die Eigenschaften des Gradienten genutzt. Dieser stellt den Vektor mit dem maximalen Zuwachs der Distanzfunktion pro Schritt dar (Konen 2021). Die korrekte Berechnung des Gradienten mehrdimensionaler Funktionen ist jedoch zu teuer, daher wird dieser mithilfe der Finite-Differenzen-Methode angenähert. Hierbei muss darauf geachtet werden, dass die Differenz möglichst minimal ist. Da allerdings mit Gleitkommazahlen gearbeitet wird, muss ein Kompromiss zwischen Präzision und Korrektheit gefunden werden, indem die Differenz proportional zur Distanz zur Kamera gewählt wird.² Die Abb. 5 zeigt Artefakte, die bei Lichtberechnungen mit ungenauen Differenzen erscheinen.

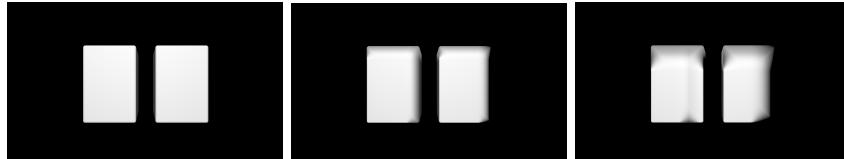


Abbildung 5: Lichtberechnung bei Verwendung von Normalen durch Finite-Differenzen-Methode mit Differenzen 0.001, 1.0 und 2.0

3.2 Texturierung

Die Farbe der Oberflächen wird dabei über eine Farbfunktion $\mathbb{R}^6 \rightarrow \mathbb{R}^3$ aus der Position und der Normalen der Oberfläche ermittelt. Dieser ergibt sich aus einem festen Grundwert, in dem verschiedene Texturen geladen werden und mit `smoothmin` und `mix` zusammengemischt werden.³ Um die verschiedenen Seiten der Boxen darzustellen, wird die Textur für jede Dimension ausgelesen und mithilfe der Normalen auf einen Quader abgebildet. Diese Technik ist ausreichend für die Beispieldarstellung, da die Boxen selber quaderförmig sind. Bei komplexerer Geometrie ließe sich diese Technik jedoch nicht gut anwenden. Die erwähnte Beispieldarstellung ist unter Abb. 7 dargestellt.

Falls die Texturen im SRGB Format vorliegen, müssen diese invers-gammakorrigiert werden, da im linearisierten Farbraum gearbeitet wird. Die finale Ausgabe wird durch `glEnable(GL_FRAMEBUFFER_SRGB)` gamma-korrigiert (Akenine-Mölle u. a. 2018; Novak 2016). Außerdem wird sie durch einen Belichtungswert und *extended reinhard tonemapping* aus einem Hochkontrastbereich in den Farbenbereich des Monitors gebracht (Salih, Malik, Saad u. a. 2012).⁴

3.3 Lichtberechnung

Auch wenn generelle globale Beleuchtungsmethoden in diesem Projekt nicht betrachtet werden, werden einige Unterpunkte berücksichtigt und vereinfacht abgebildet. Hierfür werden verschiedene Materialien verwendet, die über eine weitere dreidimensionale Funktion aus der Oberflächenposition ausgewählt werden.

²<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.gls1#L177>

³<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.gls1#L236>

⁴<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.gls1#L358>

Eine Materialfunktion bestimmt dabei den weiteren Verlauf des Strahls. Sie kann die Strahlrichtung und die Fragmentfarbe verändern. Diffuse Materialien beeinflussen dabei den Strahl direkt. Beispielsweise kann ein spiegelndes Material die Strahlrichtung an der Normalen entlang spiegeln und der Fragmentfarbe noch einen Beitrag durch diffuse Reflexion hinzufügen.

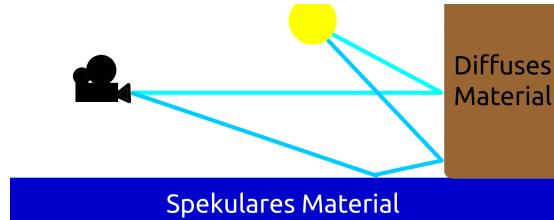


Abbildung 6: Lichtpfad von zwei Strahlen

Der Renderingalgorithmus wird durch die von Heckbert (1990) beschriebende Lichtpfadnotation wie folgt beschrieben, wobei L die Lichtquelle, $D?$ ein oder kein diffuses Material, S^* beliebig viele spekulare Materialien und E das Auge beschreibt.

$$LD?S * E$$

In dem Projekt sind zwei Materialfunktionen implementiert, die den Boden⁵ und die Boxen⁶ darstellen. Die Boxen beenden hierbei den Strahl, während der Boden reflektierend wirkt, wie in Abb. 8 zu sehen ist.

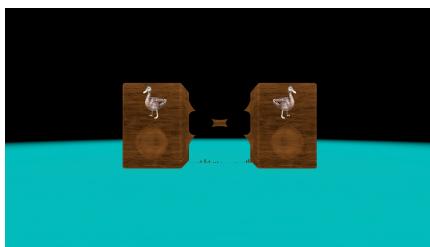


Abbildung 7: texturierte Szene

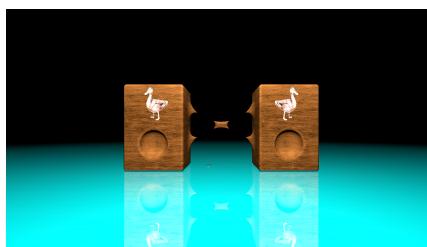


Abbildung 8: belichtete Szene

3.4 Schatten

Die Methoden aus Abschnitt 2.1 werden auch verwendet, um eine Schattenberechnung vorzunehmen. Hierbei wird ein Strahl von der Oberfläche in Richtung der Lichtquelle geschossen. Falls die Distanz zur Szene kleiner als die Distanz zur Lichtquelle ist, liegt die Oberfläche im Schatten. Dies wird über einen zwischen 0 und 1 liegenden Schattenwert abgebildet, welcher in Abb. 9 dargestellt

⁵<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.glsl#L285>

⁶<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.glsl#L278>

ist. Bei einem direkten Schatten liegt der Wert bei 1, während ein Punkt, der keinem Schatten ausgesetzt ist, einen Schattenwert von 0 hat.

Auch weiche Schatten lassen sich hiermit annähern, wie in Abb. 10 zu sehen ist. Hierfür wird der Schattenwert für einen Oberflächenpunkt auch erhöht, wenn der Strahl zur Lichtquelle nah an Szenenelementen vorbeiführt (Quilez 2010).

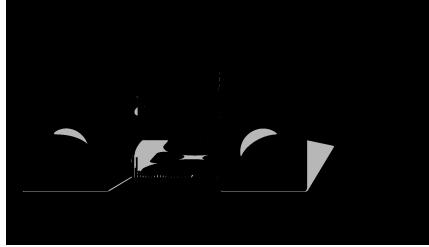


Abbildung 9: Wert von harten Schatten

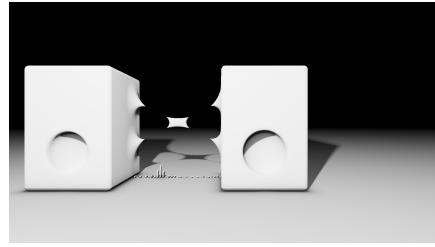


Abbildung 10: weich schattierte Szene

Das normale *Spheretracing* findet allerdings nicht zwingend die minimale Distanz eines Strahl zu der Szene zwischen zwei Schritten, falls diese nicht direkt auf dem Strahl liegt. Um Artefakte aufgrund dieses Fehlers zu vermeiden, wird dabei nicht die direkte Distanz verwendet. Stattdessen wird die vermutliche minimale Distanz beim Treffpunkt von 2 folgenden Punkten über Triangulierung berechnet (cf. Abb. 11) Hier ist der grüne Kreis die aktuelle und der rote die vergangene Szenedistanz, während in gelb die vermutlich reelle Distanz zur Szene abgebildet ist. (Aaltonen 2018, slides 38–43)⁷

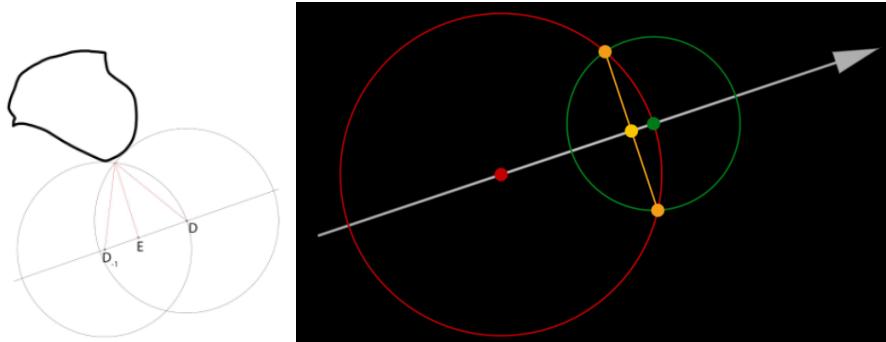


Abbildung 11: Durch Triangulation geschätzte minimale Distanz zweier Schritte.

Quilez 2010

Falls bei der Oberflächenberechnung genaue harte Kanten gewünscht sind, führt dies zu Artefakten (cf. Abb. 12).

⁷<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.glsl#L212>

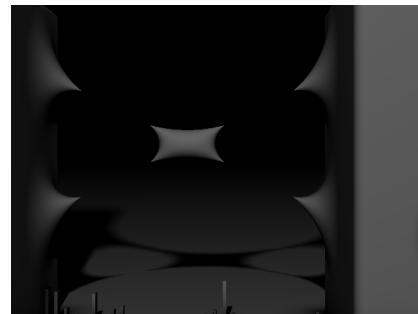


Abbildung 12: Fehler durch Unstimmigkeiten zwischen der Berechnung weicher Schatten und der Szenendistanz. Hier wirft nicht existierende Geometrie Schatten.

3.5 Umgebungsverdeckung

Es wird Umgebungsverdeckung verwendet, um die Stärke der Umgebungsbeleuchtung eines Fragmentes zu bestimmen (cf. Abb. 13).

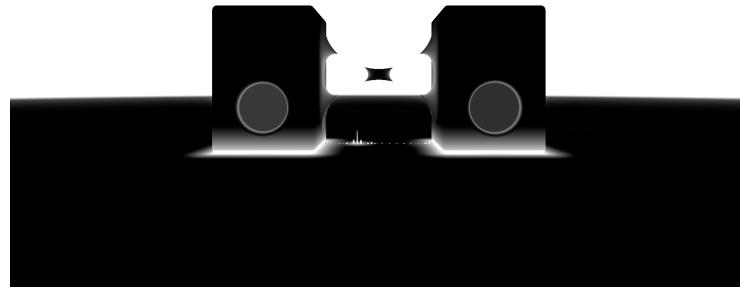


Abbildung 13: Wert der Umgebungsverdeckung in der Beispielszene

Für die Berechnung wird ein Strahl von der Oberfläche in Richtung der Normale geschossen. Es wird ein Punkt p auf diesem Strahl nahe der Oberfläche ausgewählt. An diesem Punkt wird die Differenz der Distanz zu dem Strahlensprung (und damit zu der Oberfläche) b und der minimalen Distanz zu der Szene c gemessen (cf. Abb. 14).

$$d_p = b_p - c_c$$

Dieser Schritt wird bis zu einer maximalen Schrittgröße n wiederholt. Dabei wird bei jedem Schritt erneut die Differenz des derzeitigen Punktes p_i bestimmt und eingerechnet. Die vorläufige Berechnung sieht wie folgt aus.

$$\sum_{i=1}^n \frac{d_{p_i}}{n}$$

Um an Kanten eine hohen Umgebungsbedeckung zu erreichen, werden Punkte, wie in Abb. 14 durch die abfallenden Rottöne visualisiert, abhängig von der Distanz zur Oberfläche einbezogen. Der Wert α beschreibt dabei den Wert der Umgebungsverdeckung und wird mit der Lichtintensität multipliziert (Quilez 2008).⁸

$$\alpha = 1 - \sum_{i=1}^n \frac{(d_{p_i})^2}{n - i}$$

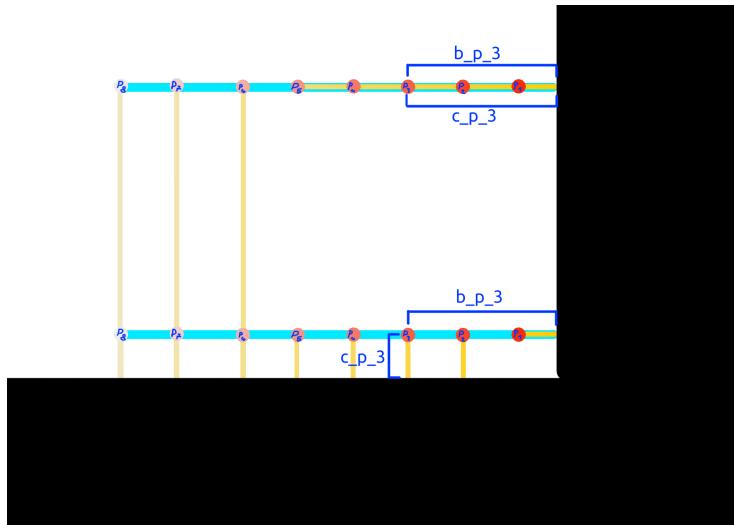


Abbildung 14: Berechnung der Umgebungsverdeckung anhand des Vergleich der Oberflächendistanz und Szenedistanz. Der untere Strahl hat dabei eine hohe Verdeckung, während der obere eine sehr niedrige hat. Die Blauen Linien bilden die Distanz zur Oberfläche ab. Die Gelben Linien bilden die Distanz zur Szene ab.

4 Hierarchische Rendermethoden

Bisher findet die Berechnung der Oberflächenfunktion auf der GPU über direkt im Shader geschriebende Ausdrücke statt.⁹ Der Hauptberechnungsanteil ist dabei die Distanzfunktion, die, wie in Abschnitt 2.1 beschrieben, für jeden Pixel mehrfach berechnet wird. Abbildung 4 beschreibt die Anzahl der Schritte und damit auch der Anzahl der Berechnungen pro Pixel. Diese bezieht sich hierbei nur auf die Positionsbestimmung und nicht die Berechnung von Schatten und Reflektionen.

Bei einer Aufnahme der Beispielszene wird die Distanzfunktion durchschnittlich 46 mal pro Pixel berechnet. Bei Verwendung eines zum Zeit dieser Arbeit

⁸<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.glsl#L201>

⁹<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.glsl#L137>

gängigen Full-HD Displays wird diese etwa $46.09375 \cdot 1920 \cdot 1080 = 95580000$ mal pro Frame berechnet und ist somit essenziell für die Ausführungszeit.

Daher wird eine Methode gesucht, um nicht die gesamte Distanzfunktion für jeden Strahl zu berechnen. Berechnungen für kleine Details werden beispielsweise erst benötigt, wenn der Schrittpunkt schon nahe an diesen ist.

4.1 Intervall Arithmetik

Duff (1992) beschreibt dabei eine mögliche Technik um mithilfe von Intervall-Arithmetik die Distanzfunktion für große Bereiche in einem Schritt zu berechnen. Um eine Szene zu rendern, wird diese in gleich große Bereiche unterteilt und das Intervall für die Teilbereiche berechnet. Falls das Intervall konstant ist, befindet sich der Teilbereich vollständig innerhalb oder außerhalb eines Objekts und muss nicht weiter berechnet werden. Andernfalls wird der Prozess rekursiv angewendet, bis dieser kleiner als der auszugebende Pixel ist.

Da das Verfahren allerdings sehr rekursiv ist, ist das Rendern auf modernen Grafikkarten nicht performant. Keeter (2020) erläutert daher ein Verfahren, welches die Rechenleistungen von Grafikkarten besser ausnutzt. Die Rekursion ist dabei auf drei Schritte beschränkt, in denen jeweils Bereiche unterteilt und mittels Intervall-Arithmetik berechnet werden. Gleichzeitig findet eine Reduktion der Distanzfunktion statt, um die Teilbereiche schneller berechnen zu können. Auch wenn dieser elementare Teil nicht in dieser Arbeit implementiert ist, ist dennoch die Vorarbeit dafür erfolgt. Diese Vorarbeit umfasst den Ansatz, Distanzfunktionen ohne Shaderkomplizierung zu interpretieren und wird im Folgenden erklärt.

4.2 Parser

Um eine Funktion zu verkürzen muss diese zuerst in einem vom Interpreter verständlichem Format vorliegen. Hierfür wurde ein Parser gebaut, der einen abstrakten Syntaxbaum aus einfachen Ausdrücken erstellt. Dieser arbeitet mit rekursiven Abstieg. Um die Rangfolge der Operatoren zu beachten, wird dabei Pratt-Parsing (Pratt 1973) verwendet.¹⁰

4.3 Tape

Das Tape beschreibt die Operatoren und die verwendeten Speicherbereiche des abstrakten Syntaxbaums. Dies liegt in der Reihenfolge des Ausführung (folglich Clause genannt) vor, so dass zuerst Knoten ohne Abhängigkeiten berechnet werden. Eine Clause wird dabei wie folgt definiert, wobei Opcode eine Operation bestimmt und output, input_A und input_B Speicheradressen sind.

```
struct clause {
    Opcode opcode;           // 1 byte
    uint8_t output;          // 1 byte
    uint8_t input_A{};       // 1 byte
    uint8_t input_B{};       // 1 byte
};
```

¹⁰<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/src/Parser.cpp>

Das Tape wird dabei von dem abstrakten Syntaxbaum rekursiv vom Root Knoten aus gebaut. Es arbeitet mit einem festem limitierten Speicherbereich, welcher vor der Berechnung zugewiesen wird. Bei Knoten mit Abhängigkeiten füllen dabei zuerst deren Kindknoten das Tape und den Speicherbereich, bevor sie sich selbst in diesen einfügen. Das Zuweisen der Speicheradressen ist unter Register Allocation Problem bekannt. Um das Problem zu lösen, werden von Kindknoten freigegebene Speicherbereiche wieder verwendet, was zu einer kompakten Speicherverwendung führt. Tabelle 1 zeigt ein erstelltes Tape von der Distanzfunktion $\sqrt{x \cdot x + y \cdot y + z \cdot z} - 1$, welche eine Kugel darstellt.

Das Tape wird in einem TexBuffer gespeichert¹¹, um in der *Fragment Shader Stage* für jeden Pixel ausgelesen und ausgeführt zu werden.¹²

opcode	input_A	input_B	output
X	-	-	slot 0
Y	-	-	slot 1
SQUARE	slot 0	-	slot 0
SQUARE	slot 1	-	slot 1
ADD	slot 0	slot 1	slot 1
SQRT	slot 1	-	slot 1
SUB	slot 1	1.0f	slot 0
SUB	0.5f	slot 1	slot 1
MAX	slot 0	slot 1	slot 1

Tabelle 1: Beispiel eines Tapes der Distanzfunktion einer Kugel. (Keeter 2020, Table 1)

4.4 Renderzeiten

Wie Keeter (2020) anmerkt und in Abb. 15 zeigt, ist die interpretierte Berechnung der SDF nur dann sinnvoll, wenn diese algorithmische Optimierung erlaubt. Um dies zu testen, wurde eine Szene aus einer Kugel und einer Ebene mit folgender Distanzfunktion gebaut.

$$d(x, y, z) = \min(y, \sqrt{(x - 1)^2 + (y - 1)^2 + (z - 6)^2} - 1)$$

Falls das Lesen der Szene aus einem Buffer geschieht, beträgt die Renderzeit eines Frames ca 113113 μs . Falls die Distanzfunktion im Shader compiliert wurde, beträgt die Renderzeit eines Frames ca 1035 μs .

¹¹<https://github.com/spookyGh0st/ip/blob/9f0523552634f3f05766b6b7382e7aeb3d62f642/src/ShaderProgram.cpp#L105>

¹²<https://github.com/spookyGh0st/ip/blob/2f45cd314b70b9108216fae12ab2768bb1064f9d/assets/shaders/frag.glsl#L123>

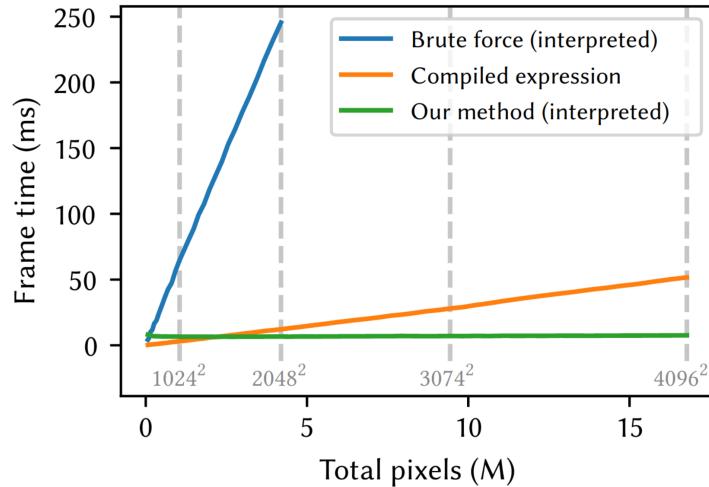


Abbildung 15: von Keeter (2020) bemessene Framezeiten eines Beispielmodells. Gemessen an einer NVIDIA GTX 1080 Ti GPU.

5 Audio

Um das Projekt zu abzurunden, wird die vorhandene Beispielsszene zu einem Audiovisualizer ausgearbeitet. Das eingelesene Audiosignal wird dabei in den verschiedenen Spuren in **Buffer** unterteilt, die die Audiowelle einer gewissen Zeit durch eine Menge von **Samples** abbilden.

Zu der Visualisierung werden alle Audiodaten des letzten Frames verwendet, die Buffergröße wird also aus der Delta Time und der Sample Rate des Audios berechnet. Hierdurch hinkt die Visualisierung der Bildausgabe etwas hinterher, was jedoch bei der verwendeten Umgebung nicht ins Gewicht fällt. Allerdings stimmt die Visualisierung nicht in Umgebungen mit höheren Latenzzeiten, wie etwa bei der Übertragung über Bluetooth.

Das Projekt hat zwei Arten der Visualisierung. Es wird zum Einen die Lautstärke der Tonspuren visualisiert, die jeweils durch

$$V = \frac{\sum_{i=0}^n |S_i|}{n} \quad (1)$$

berechnet wird, wobei S den Samplebuffer und n die Größe des Samplebuffers seit letzten Frame darstellt.

Zum Anderen wird ein Histogramm wie in Abb. 16 über mehrere Tonhöhen ausgegeben, die mittels des *Fast Fourier Transforms* bestimmt werden.

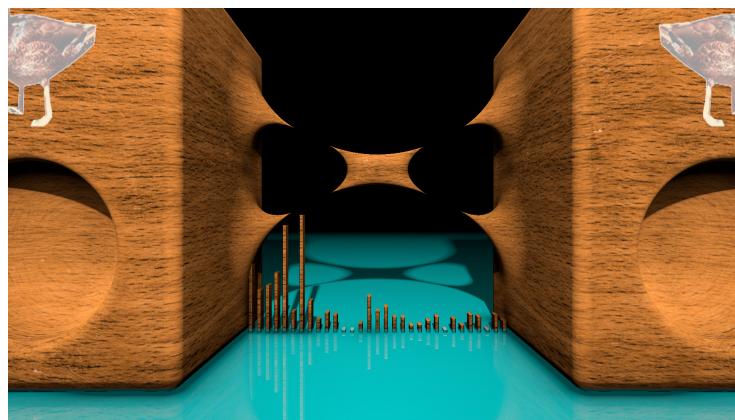


Abbildung 16: Detailansicht des Histogramm

6 Fazit und Ausblick

Eine Aufnahme der finalen Arbeit ist unter https://youtu.be/uRfuHaP_5FA zu sehen. Die Verwendung impliziter Oberflächen für die Grafikdarstellung erscheint als valide Technik. Allerdings sind die implementierten Methoden noch sehr Performance intensiv und können nur begrenzt komplexe Szenen darstellen. Die Arbeit von Keeter (2020) gibt einen guten Ausblick darauf, diese Limitierung aufzuheben. Persönlich wurde durch diese Arbeit viel gelernt und ich freue mich auf die Entwicklung der Forschung.

Literatur

- Advanced Graphics Techniques Tutorial: GPU-Based Clay Simulation and Ray-Tracing Tech in Claybook* (März 2018). Game Developers Conference. URL: <https://www.gdcvault.com/play/1025316/Advanced-Graphics-Techniques-Tutorial-GPU>.
- Abt, Bastian (Feb. 2022). *Mariana_rec*. URL: https://www.youtube.com/watch?v=uRfuHaP_5FA.
- Akenine-Mölle, T. u. a. (2018). *Real-Time Rendering*. 4th Edition. CRC Press, S. 160–166. ISBN: 9781138627000.
- Blinn, James F (1977). „Models of light reflection for computer synthesized pictures“. In: S. 192–198.
- Audio in standard c++* (2019). C++Now 2019.
- Duff, Tom (1992). „Interval arithmetic recursive subdivision for implicit functions and constructive solid geometry“. In: *ACM SIGGRAPH computer graphics* 26.2, S. 131–138.
- Först, Vincent (März 2021). *Demoszene wird Immaterielles Kulturerbe*. URL: <https://netzpolitik.org/2021/digitale-kunst-demoszene-wird-immaterielles-kulturerbe/>.
- Hart, John C (1996). „Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces“. In: *The Visual Computer* 12.10, S. 527–545.
- Heckbert, Paul S (1990). „Adaptive radiosity textures for bidirectional ray tracing“. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, S. 145–154.
- Keeter, Matthew J (2020). „Massively parallel rendering of complex closed-form implicit surfaces“. In: *ACM Transactions on Graphics (TOG)* 39.4, S. 141–1.
- Konen, Wolfgang (2021). *Der Gradient: Wo bitte geht's nach oben?* URL: <https://www.gm.fh-koeln.de/~konen/Mathe2-SS/ZD2-Kap08.pdf>.
- Mitchell, Don P (1990). „Robust ray intersection with interval arithmetic“. In: *Proceedings of Graphics Interface*. Bd. 90, S. 68–74.
- Moore, Ramon E, R Baker Kearfott und Michael J Cloud (2009). *Introduction to interval analysis*. SIAM.
- Novak, John (Sep. 2016). *What every coder should know about gamma*. Last accessed 18 Jan 2022, backup: <https://web.archive.org/web/20211229231200/https://blog.johnnovak.net/2016/09/21/what-every-coder-should-know-about-gamma/>. URL: <https://blog.johnnovak.net/2016/09/21/what-every-coder-should-know-about-gamma>.
- Pratt, Vaughan R (1973). „Top down operator precedence“. In: *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, S. 41–51.
- (2008). nvscene, S. 48–54. URL: <https://iquilezles.org/www/material/nvscene2008/rwtt.pdf>.
- Quilez, Inigo (Juni 2010). *soft shadows in raymarched SDFs - 2010*. Last accessed 09 Jan 2022, backup: <https://web.archive.org/web/20220109160418/https://www.iquilezles.org/www/articles/rmshadows/rmshadows.htm>. URL: <https://www.iquilezles.org/www/articles/rmshadows/rmshadows.htm>.
- (Aug. 2011). *distance functions*. Last accessed 09 Jan 2022, backup: <https://web.archive.org/web/20220125224041/https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm>. URL: <https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm>.

- //www.iquipilezles.org/www/articles/distfunctions/distfunctions.htm.
- Quilez, Inigo (Okt. 2013). *smooth minimum*. Last accessed 09 Jan 2022, backup: <https://web.archive.org/web/20211219111428/https://www.iquipilezles.org/www/articles/smin/smin.htm>. URL: <https://www.iquipilezles.org/www/articles/smin/smin.htm>.
- (2020). *Interior Distance*. Last accessed 09 Jan 2022. URL: <https://www.iquipilezles.org/www/articles/interiordistance/interiordistance.htm>.
- Raposo, Adriano N und Abel JP Gomes (2019). „Pi-surfaces: products of implicit surfaces towards constructive composition of 3D objects“. In: *arXiv preprint arXiv:1906.06751*.
- Ricci, Antonio (1973). „A constructive geometry for computer graphics“. In: *The Computer Journal* 16.2, S. 157–160.
- Salih, Yasir, Aamir S Malik, Naufal Saad u. a. (2012). „Tone mapping of HDR images: A review“. In: *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*. Bd. 1. IEEE, S. 368–373.
- Sederberg, Thomas W und Scott R Parry (1986). „Free-form deformation of solid geometric models“. In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, S. 151–160.

7 Anhang



Abbildung 17: Mariana Duck Texture



Abbildung 18: Wood Texture

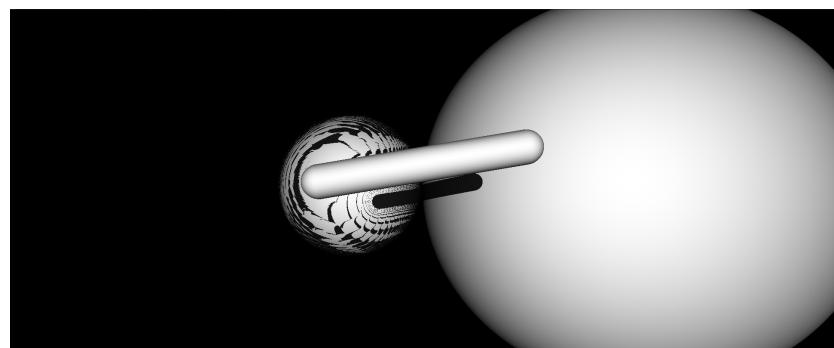


Abbildung 19: Artefakte bei Sphere Tracing von sehr weit entfernten Objekten durch floating point ungenauigkeiten