

MEDIENINFORMATIK / GRAPHICS & GEOMETRIC COMPUTING

Praktikum Computergrafik

Sommersemester 2023

**Visualisierung von Mikrokosmen und
Fraktalen mit dem Lenia-Algorithmus und
dem Ray-Marching-Verfahren**

August 2023

Visualisierung von Mikrokosmen und Fraktalen mit dem Lenia-Algorithmus und dem Ray-Marching-Verfahren

Inhaltsverzeichnis

Einführung	1
1 Lenia	1
1.1 Conway's Game of Life	1
1.2 Lenia	1
1.3 Implementierung von Lenia	2
2 Ray-Marching	3
2.1 Darstellung von Objekten	3
2.2 Ray-Marching-Algorithmus	3
2.3 Licht und Schatten	3
2.4 Fraktalstrukturen	4
2.5 Interaktion mit dem Lenia-Mesh	5
3 Fazit	5

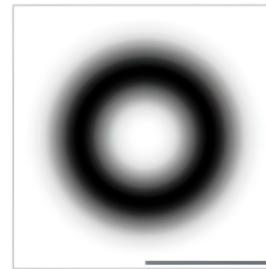
Einführung

In diesem Bericht werden verschiedene Methoden zur Erzeugung von biologisch inspirierten Visualisierungen vorgestellt und erläutert, was bei deren Implementierung beachtet werden muss. Zunächst wird ein Überblick über Conway's Game of Life gegeben, das als Grundlage für den Lenia-Algorithmus dient. Anschließend wird die Fraktal-Generierung mittels Ray-Marching erläutert. Das Ziel ist es, die abstrakten Modelle zu implementieren und dabei visuell ansprechende Ergebnisse zu erzielen, die nahezu in Echtzeit laufen.

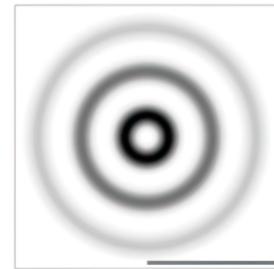
1. Lenia

1.1 Conway's Game of Life

Conway's Game of Life ist ein von John Conway entwickeltes Spiel, das auf einem zweidimensionalen Gitter basiert und einen Mikrokosmos simuliert. Es verwendet einen zweidimensionalen, zellulären Automaten, bei dem die Zellen zu Beginn entweder lebendig oder tot sein können. Der Folgezustand jeder Zelle wird durch die Anzahl der lebendigen Nachbarzellen in der Moore-Nachbarschaft bestimmt. Eine lebendige Zelle bleibt im nächsten Schritt lebendig, wenn sie zwei oder drei lebendige Nachbarn hat. Eine tote Zelle wird im nächsten Schritt lebendig, wenn sie genau drei lebendige Nachbarn hat. In allen anderen Fällen wird die Zelle im nächsten Schritt tot sein. Dies führt zu einem geschlossenen System, das sich selbstständig weiterentwickelt [1].



(a) Kernel-Shell



(b) Kernel-Skeleton

Abbildung 1. [2]

1.2 Lenia

Der Lenia-Algorithmus gehört ebenfalls zur Familie der zellulären Automaten, unterscheidet sich jedoch von Conway's Game of Life durch sein kontinuierliches System. Eine Zelle kann nicht nur lebendig oder tot sein, sondern auch einen beliebigen Zustand zwischen 0 und 1 annehmen. Die Moore-Nachbarschaft wurde durch eine von der Distanz abhängige Nachbarschaft ersetzt, die es ermöglicht, die Nachbarschaft zu erweitern und zu gewichten. Die Regeln von Conway's Game of Life wurden durch eine Kernel- und eine Growth-Funktion ersetzt, die die Veränderung der Zellen im nächsten Schritt bestimmen. Zuerst wird mithilfe eines Radius eine Nachbarschaft um die Zelle gebildet. Die Zustände der Zellen in der Nachbarschaft werden dann mit der Kernel-Funktion gewichtet und anschließend mit der Growth-Funktion zu einem neuen Wert zusammengefasst. Dieser wird gewichtet auf die Zelle addiert und ergibt den neuen Zustand der Zelle. Der Kernel besteht aus zwei Teilen: der *Kernel-Shell* und dem *Kernel-Skeleton*. Für die Kernel-Shell kann beispielsweise die Exponentialfunktion $K_C(r) = \exp\left(4 - \frac{4}{4r(1-r)}\right)$ gewählt werden, welche wie Abbildung 1a aussieht und die Nachbarschaft gewichtet. Hierbei ist zu beachten, dass der Radius r auf einem Bereich zwischen 0 und 1 normiert ist. Soll dies nun auf einem Mesh stattfinden, so bietet es sich an, zuerst die durchschnittliche Kantenlänge der Faces zu bestimmen und ein Vielfaches davon als Norm für den Radius zu wählen. Des Weiteren lässt sich die Distanz zwischen zwei Faces eines Meshes entweder durch die euklidische Distanz oder durch die Distanz auf der Oberfläche bestimmen. Die geodätische Distanz ist die kürzeste Distanz zwischen zwei Punkten auf einer Oberfläche. Es gibt verschiedene Algorithmen zur Be-

stimmung dieser Distanz, die sich in Genauigkeit und Laufzeit unterscheiden. Die meisten davon sind jedoch darauf ausgelegt, die Distanz zwischen zwei Vertices und nicht zwischen zwei Faces zu bestimmen. Wird nun zuerst das duale Mesh bestimmt, so wird das Zentrum eines Faces zu einem Vertex und die Kanten zu Faces. Die Distanz zwischen zwei Faces ist dann die Distanz zwischen den zugehörigen Vertices. Alle Faces, die innerhalb des Radius liegen, werden dann in die Nachbarschaft aufgenommen. Zusätzlich zur Kernel-Shell können noch Peaks hinzugefügt werden, welche in das Kernel-Skeleton einfließen. Die Peaks sind dabei in einer Liste gespeichert $\beta = \{\beta_0, \dots, \beta_{n-1}\} \in [0, 1]^n$ und das Kernel-Skeleton ergibt sich wie folgt:

$$K_S(r) = \beta_{\lfloor n \cdot r \rfloor} \cdot K_C(n \cdot r \bmod 1)$$

n gibt dabei die Anzahl der Peaks an, r ist die zwischen 0 und 1 normierte Distanz und $\bmod 1$ gibt den Nachkommanteil von $n \cdot r$ an. Die für β verwendeten Werte geben dabei die Gewichtung der Peaks an. Abbildung 1b zeigt ein Beispiel für ein Kernel-Skeleton mit drei verschiedenen Peaks. Damit der Kernel vollständig ist, muss dieser nun noch normiert werden. Dies geschieht für eine Zelle wie folgt:

$$N := \text{Menge aller Nachbarn der betrachteten Zelle}$$

$$n := \text{ein Nachbar der betrachteten Zelle}$$

$$\text{dist}(n) := \text{Distanz zwischen betrachteter Zelle und Zelle } n$$

$$\text{area}(n) := \text{Fläche von Zelle } n$$

$$K(n) := \frac{K_S(\text{dist}(n))}{\sum_{n \in N} (K_S(\text{dist}(n)) \cdot \text{area}(n))}$$

Der Nenner sorgt dabei für die Normierung des Kernels, was durch die noch folgenden Gleichungen deutlich wird. Ist nun der Kernel vollständig, so lässt sich ein Zwischenwert für die Zelle in Abhängigkeit von der Nachbarschaft bestimmen:

$$x := \text{betrachtete Zelle}$$

$$A^t(x) := \text{aktueller Zustand der Zelle } x$$

$$U(x) = \sum_{n \in N} (K(n) \cdot A^t(n) \cdot \text{area}(n))$$

Nachdem der Zwischenwert für die Zelle bestimmt wurde, wird dieser durch die Growth-Funktion zu einem neuen Wert zusammengefasst. Die Growth-Funktion ist eine beliebige, kontinuierliche Funktion, die den Zwischenwert, welcher zwischen 0 und 1 liegt, auf einen neuen Wert zwischen -1 und 1 abbildet. Hier kann beispielsweise eine Exponentialfunktion $G(u) = 2 \cdot \exp\left(-\frac{(u-\mu)^2}{2 \cdot \sigma^2}\right) - 1$ gewählt werden. Die Parameter μ und σ können dabei frei angepasst werden. μ gibt das Maximum und σ die Breite der Funktion an. Abbildung 2 zeigt ein Beispiel für verschiedene Parameter. Ist σ ein kleiner Wert, so werden mehr Werte auf -1 abgebildet, wodurch die Zellen schneller sterben. Nachdem die Growth-Funktion bestimmt wurde, folgt die Aktualisierung der Zelle. Dafür wird

der neue Wert der Zelle mit einem Gewicht auf den alten Wert addiert und zwischen 0 und 1 begrenzt.

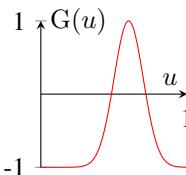
$$A^{t+1}(x) := \text{neuer Zustand der Zelle } x$$

$$g := \text{Gewichtungsfaktor}$$

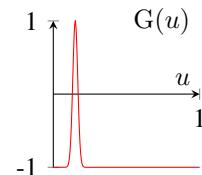
$$\text{clip}(u, v, w) := \max(u, \min(v, w))$$

$$A^{t+1}(x) := \text{clip}(A^{t+1}(x) + g \cdot G(U(x)), 0, 1)$$

Damit ist die Aktualisierung einer Zelle für einen Zeitschritt abgeschlossen. Der Gewichtungsfaktor $g \in (0, 1]$ und der Zeitschritt $t + 1$ bestimmen dabei die zeitliche Auflösung des Systems. Je kleiner der Gewichtungsfaktor, desto langsamer und flüssiger entwickelt sich das System.



(a) $\mu = 0.6, \sigma = 0.1$



(b) $\mu = 0.15, \sigma = 0.017$

Abbildung 2. Exponentialfunktion als Growth-Funktion mit verschiedenen Parametern

1.3 Implementierung von Lenia

Ausgehend davon, dass der Lenia-Algorithmus auf einem statischen Mesh ausgeführt wird, können einige Dinge vorberechnet werden. Eine naive Implementierung würde für jeden Zeitschritt jede Zelle durchgehen, die Nachbarschaft bestimmen und die benötigten Werte berechnen. Wenn sich das Mesh und die für den Algorithmus verwendeten Parameter nicht ändern, kann die Nachbarschaft für jede Zelle einmal beim Laden des Meshes bestimmt und danach wiederverwendet werden. Ebenfalls fällt auf, dass der Wert des Kernels $K(n)$ von der Distanz zwischen der betrachteten Zelle und der Zelle n abhängt. Wenn sich die Nachbarschaft einer Zelle nicht ändert, kann der Wert des Kernels für alle Nachbarn einer Zelle einmal berechnet und anschließend wiederverwendet werden. Hierbei ist zu beachten, dass eine Zelle nicht nur einen Kernel-Wert hat. Der Kernel-Wert hängt von der Nachbarschaftsrelation ab, weshalb es sinnvoll ist, ihn mit der zuvor erwähnten Nachbarschaftsrelation zu speichern. Darüber hinaus bleibt die Fläche $\text{area}(n)$ einer Zelle auf einem statischen Mesh konstant, so dass sie ebenfalls vorberechnet werden kann. Die Kernel-Funktion K_S und die Funktion U enthalten beide eine Summe mit derselben Laufvariable und demselben Endwert, sodass sie in der Implementierung in einer Schleife zusammengefasst werden können. Wenn all diese Werte vorberechnet werden, muss bei der Aktualisierung des Zustands einer Zelle nur der aktuelle Zustand abgefragt und mit bereits berechneten Werten multipliziert werden, um den Kernel-Wert zu erhalten. Dieser wird dann in die Growth-Funktion eingesetzt und der neue Zustand der Zelle bestimmt. Bei der Aktualisierung des Zustands des gesamten Systems

wird jede Zelle durchlaufen und anhand der zuvor bestimmten Nachbarschaft der neue Zustand bestimmt. Da der nächste Zustand immer nur vom aktuellen Zustand abhängt, kann dieser Prozess parallelisiert werden. Der Wert jeder Zelle kann unabhängig von den anderen Zellen bestimmt werden. Dadurch lässt sich die Berechnung des nächsten Zustands auf mehrere Threads aufteilen und beschleunigen.

2. Ray-Marching

Eine weitere Möglichkeit, biologisch inspirierte Strukturen zu erzeugen, sind Fraktale. Fraktale sind selbstähnliche Strukturen, die immer feiner aufgelöst werden können. Es ist möglich, diese Fraktale als Meshes zu modellieren und auf herkömmliche Weise zu rendern. Allerdings ist dies nur bis zu einem gewissen Punkt praktikabel. Ein alternatives Verfahren zum Rendern der Strukturen ist das Ray-Marching, bei welchem die Strukturen nicht als Meshes, sondern als implizite Funktionen dargestellt werden [3].



Abbildung 3. [4]

2.1 Darstellung von Objekten

Die Darstellung von Objekten erfolgt mittels *Signed Distance Functions* (SDF). Diese Funktionen geben für einen bestimmten Punkt im Raum den Abstand zur Oberfläche eines Objekts an. Die Funktionen können sowohl positive als auch negative Werte zurückgeben, je nachdem, ob sich der Punkt innerhalb oder außerhalb der Struktur befindet. Ein Beispiel für eine primitive Funktion ist die einer Kugel im Ursprung mit einem Radius von 1. Diese kann als $\|p\|_2 - r$ beschrieben werden, wobei $\|p\|_2$ der euklidische Abstand des Punktes p zum Ursprung und r der Radius der Kugel ist. Wenn p innerhalb der Kugel liegt, ist der Abstand negativ, wenn p außerhalb der Kugel liegt, ist der Abstand positiv und wenn p genau auf der Oberfläche der Kugel liegt, ist der Abstand 0. Für eine Vielzahl von primitiven und komplexeren Strukturen können solche SDFs gefunden werden [5]. Befinden sich nun zwei oder mehr Objekte im Raum, so können diese durch verschiedene Operationen miteinander verknüpft werden. Wenn das Minimum der beiden Funktionen gebildet wird, ergibt sich eine neue Funktion, die den Abstand zum nächstgelegenen der beiden Objekte angibt. Beim Maximum der beiden Funktionen, ergibt sich eine Funktion, die den Schnitt der beiden Objekte beschreibt. Es ist auch möglich, ein Objekt von dem anderen zu subtrahieren, indem das Maximum eines Objekts zusammen mit dem negativen Wert des anderen Objekts, das von dem ersten Objekt abgezogen werden soll, gebildet wird. Abbildung 3 zeigt die Vereinigung und den Schnitt eines Würfels mit einer Kugel sowie das Ergebnis, wenn die Kugel von dem Würfel subtrahiert wird.

2.2 Ray-Marching-Algorithmus

Zum Darstellen der Objekte wird ein Ray-Marching-Algorithmus verwendet. Dabei wird für jedes Pixel des Bildes ein Strahl von der Kamera in die Szene geschickt. Mit Hilfe der SDF kann der Abstand zum nächsten Objekt bestimmt werden. Obwohl nicht bekannt ist, wo sich das nächstgelegene Objekt befindet, kann sichergestellt werden, dass der Strahl nicht zu weit in die Szene hineingeschickt wird. Der Strahl läuft dann um den berechneten Abstand weiter in die vorgegebene Richtung und von diesem Punkt aus wird erneut der Abstand zum nächsten Objekt berechnet. Dieser Vorgang wird iterativ wiederholt, bis entweder ein oberes Limit an Iterationen erreicht ist, eine maximal zulässige Distanz überschritten wurde oder der Abstand zum nächsten Objekt kleiner als ein vorgegebenes Minimum ist. Wenn dieses Minimum erreicht ist, wird angenommen, dass der Strahl auf ein Objekt trifft. Wenn das obere Limit der Iterationen erhöht oder das Minimum verringert wird, erhöht sich die Genauigkeit des Algorithmus, wodurch die Objekte höher aufgelöst werden, aber auch die Laufzeit steigt.

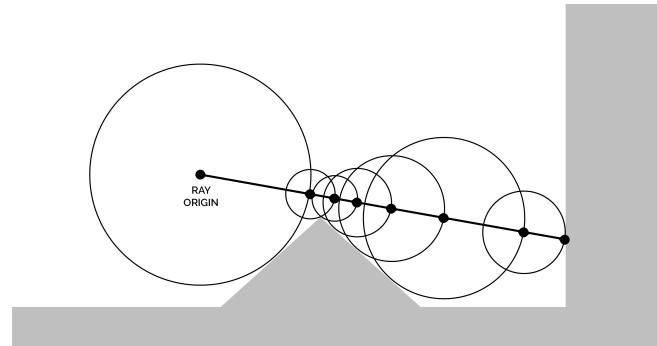


Abbildung 4. Visualisierung des Ray-Marching-Algorithmus. [6]

2.3 Licht und Schatten

Trifft nun ein Strahl auf ein Objekt, so kann für diesen Punkt ein Beleuchtungsmodell ausgewertet werden. Wird dafür beispielsweise das Phong-Beleuchtungsmodell verwendet, so wird dafür die Normale an dem Punkt benötigt. Für einfache Objekte wie Kugeln oder Flächen kann diese Normale unkompliziert bestimmt werden, für komplexere Strukturen kann dies jedoch schwierig sein. Die Normale eines Punktes kann mit dem Gradienten der SDF an diesem Punkt bestimmt werden. Dies kann numerisch approximiert werden, indem die SDF um den Punkt herum in alle drei Richtungen mit einem kleinen Abstand ermittelt und der Abstand zum Punkt berechnet wird. Dies entspricht der Bestimmung des Gradienten der SDF an diesem Punkt, womit die Normale an diesem Punkt bestimmt werden kann. Ist die Normale an dem Punkt bekannt, so kann das bekannte Phong-Beleuchtungsmodell verwendet werden, um die Beleuchtung an dem Punkt auszuwerten, wie in Abbildung 5b zu sehen ist.

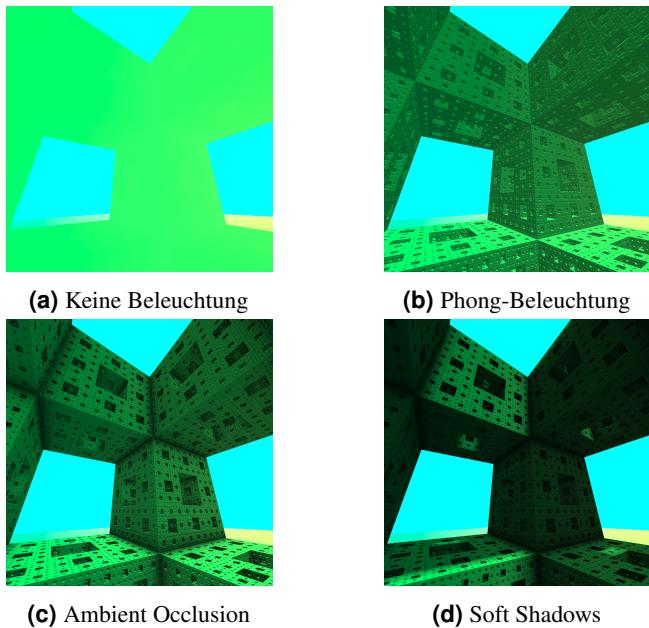


Abbildung 5. Inkrementelles Hinzufügen von Beleuchtungseffekten

Mithilfe der Normalen lässt sich ebenfalls Ambient Occlusion bestimmen. Dazu wird von einem Oberflächenpunkt aus entlang der Richtung der Normale ein Strahl in die Szene geschickt und dabei der Abstand zum nächsten Objekt bestimmt. Dies geschieht für eine feste Anzahl von Schritten, wobei in jedem Schritt die Weite des Strahls vergrößert wird. Dabei werden die ermittelten Entfernungswerte und die vom Strahl zurückgelegte Distanz aufsummiert. Wenn kein anderes Objekt in der Nähe liegt, ist die Summe der Objektentfernungswerte S gleich der Summe der zurückgelegten Distanzen M , da die SDF in jedem Schritt den Abstand zum Ausgangspunkt zurückgeben würde. Wenn ein anderes Objekt in der Nähe ist, wäre S kleiner als M . $S \div M$ ergibt dann einen Wert zwischen 0 und 1, der als Ambient Occlusion-Wert verwendet werden kann, indem er mit dem Farbwert des Punktes multipliziert wird. Dieses Verfahren kann verfeinert werden, indem weiter entfernte Distanzen weniger stark gewichtet werden [7]. Das Ergebnis ist in Abbildung 5c dargestellt.

Eine einfache Möglichkeit Schatten zu erzeugen besteht darin, von jedem Punkt auf der Oberfläche einen Strahl zur Lichtquelle zu senden und zu überprüfen, ob dieser Strahl auf ein Objekt trifft. Wenn der Strahl auf ein Objekt trifft, liegt der Punkt im Schatten und es wird ein dunklerer Farbwert verwendet. Dieses Verfahren erzeugt scharfe Schatten, die jedoch nicht immer erwünscht sind. Wenn ein Strahl auf dem Weg zwischen dem Punkt und der Lichtquelle ein Objekt nur knapp verfehlt, könnte dieser im Halbschatten liegen, um ein weicheres Schattenbild zu erzeugen. Für jeden Schritt der Ray-Marching-Iteration zwischen Punkt und Lichtquelle kann ein Halbschatten-Faktor bestimmt werden. Dieser ergibt sich aus dem Wert der SDF des aktuellen Punktes der Iteration, der ins Verhältnis zur Distanz zum Ausgangspunkt gesetzt

wird, sodass weiter entfernte Objekte weniger stark gewichtet werden. Von den bestimmten Halbschatten-Faktoren entlang des Strahls wird dann der kleinste Wert verwendet, um den Farbwert des Punktes zu bestimmen. Aufgrund der geringeren Gewichtung der Werte für weiter entfernte Objekte entsteht außerdem der Effekt, dass die Schatten weicher werden, je weiter sie von dem Objekt entfernt sind und umgekehrt [8]. Dieser Effekt ist in Abbildung 6b dargestellt.

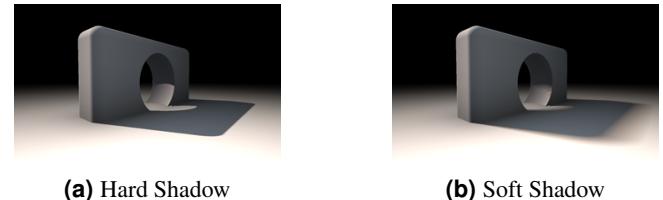


Abbildung 6. [8]

2.4 Fraktalstrukturen

Um fraktalartige Strukturen zu erstellen, können SDFs verwendet werden, die sich unendlich oft wiederholen. Eine Kugel, die mit einer SDF dargestellt wird und deren untere linke Ecke im Ursprung liegt und einen Radius von 1 hat, ist in Abbildung 7a dargestellt. Die SDF nimmt einen Punkt im Raum als Eingabe und gibt den Abstand zu der Oberfläche der Kugel zurück. Dieser Punkt im Raum ist der aktuelle Punkt der Ray-Marching-Iteration. Wenn auf die x-, y- und z-Koordinate dieses Punktes der Modulo-Operator mit beispielsweise 5 angewendet wird, liegen die jeweiligen Werte immer zwischen 0 und 5. Das bedeutet, dass wenn der Strahl auf der x-Achse nach rechts die 5 überschreiten würde, er dies nicht tut und stattdessen von links wieder bei 0 anfängt. Geschieht dies für alle Achsen, ergibt sich ein impliziter Kasten mit einer Kantenlänge von 5. Alle Objekte innerhalb dieses Kastens werden aufgrund der Modulo-Operation unendlich oft in alle Richtungen wiederholt. Abbildung 7b zeigt, wie dies für eine Kugel ausssehen kann. Wenn dies nur in einer Achsenrichtung geschehen soll, kann der Modulo-Operator nur auf eine Achse angewendet werden.

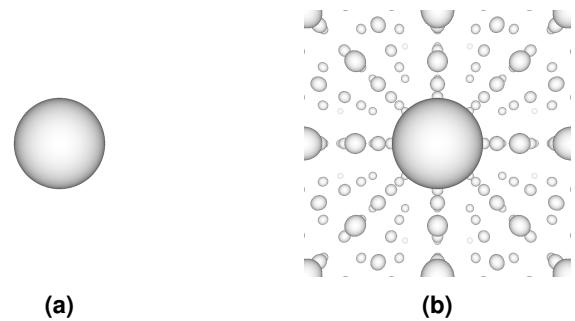


Abbildung 7

Der *Menger-Sponge* ist ein Beispiel für eine fraktale Struktur, die sich mit den vorgestellten Methoden dargestellt lässt. Dazu wird zunächst mit einem Würfel begonnen. Von diesem

wird dann ein Kreuz, wie in Abbildung 8e zu sehen, mithilfe der in Kapitel 2.1 beschriebenen Operationen abgezogen. Anschließend wird das Kreuz auf 1/3 seiner vorherigen Größe skaliert und mit dem zuvor erklärten Verfahren in alle Richtungen unendlich oft wiederholt und ebenfalls von dem Würfel abgezogen. Dieses Verfahren wird nun iterativ wiederholt, bis der Menger-Sponge die gewünschte Auflösung erreicht hat. Abbildung 8 zeigt dieses iterative Verfahren. Dabei ist in grün die aktuelle Iteration zu sehen und in rot die Kreuze, die abgezogen werden, um in den nächsten Iterationsschritt zu gelangen. Zur Veranschaulichung sind diese auf den Bereich des Würfels beschränkt. Der dadurch entstehende Menger-Sponge ist auch wieder eine SDF, da das beschriebene Verfahren nur das Maximum von zwei SDFs bildet und somit selbst auch wieder eine SDF ist. Es wäre daher beispielsweise möglich, unendlich viele Menger-Sponges nebeneinander zu platzieren oder einen solchen von anderen Objekten abzuziehen, beziehungsweise andere Objekte von diesem abzuziehen, um so komplexere Strukturen zu erzeugen. Da eine Signed Distance Function nicht nur als mathematische Funktion, sondern auch als eine aus Programmiersprachen bekannte Funktion verstanden werden kann, lassen sich damit viele weitere interessante Strukturen erzeugen.

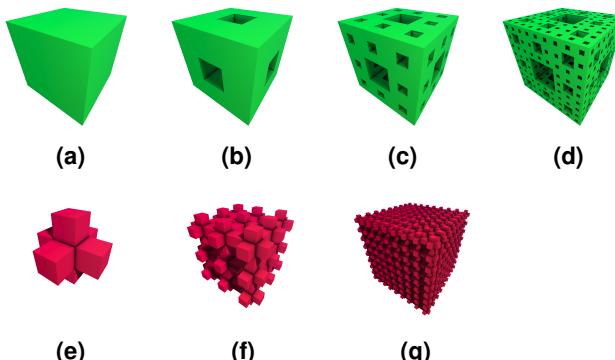


Abbildung 8. Iterativ entstehender Menger-Sponge.

2.5 Interaktion mit dem Lenia-Mesh

Der in Kapitel 1 besprochene Lenia-Algorithmus kann auf Mesh-Oberflächen angewendet werden. Das in diesem Kapitel vorgestellte Ray-Marching-Verfahren stellt jedoch keine Mesh-Objekte dar. Das in Abbildung 9 gezeigte Ergebnis ergibt sich dadurch, dass es einen Shader für das Ray-Marching und einen für das Mesh-Objekt gibt. Mit dem Ray-Marching-Shader kann eine Environment-Map erzeugt werden, die im Mesh-Shader für die Reflexionen verwendet wird. Wenn der Depth-Buffer beider Fragment-Shader entsprechend angepasst wird, ist es auch möglich, dass ein Objekt aus dem einen Shader das Objekt aus dem anderen Shader verdeckt. Auf die bei der Implementierung zu beachtenden Details wird in diesem Bericht nicht weiter eingegangen.

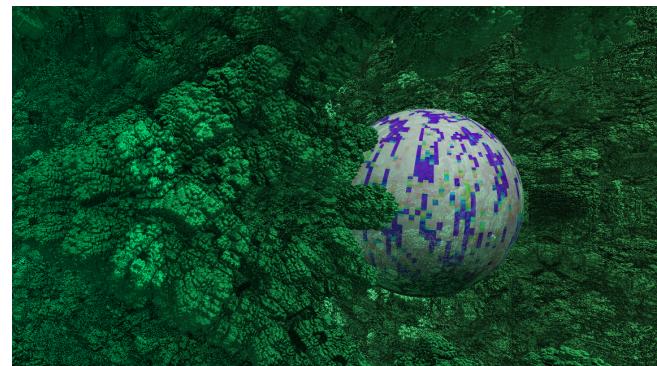


Abbildung 9. Überschneidung von Mesh-Objekt mit Ray-Marching Objekt und Reflexion.

3. Fazit

Zusammenfassend lässt sich sagen, dass die vorgestellten Methoden vielfältige Möglichkeiten bieten, um von der Biologie inspirierte Visualisierungen zu erzeugen und dabei weiterhin Raum für kreative Ideen lassen. Die vorgestellten Algorithmen laufen auf entsprechend leistungsfähiger Hardware in Echtzeit und können durch Veränderung der Parameter in Echtzeit angepasst werden.

Literatur

- [1] Wikipedia. *Conway's Game of Life*. URL: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life (besucht am 2023-08-23).
- [2] Bert Wang-Chak Chan. *Lenia - Biology of Artificial Life*. 2018. arXiv: 1812.05433 [nlin.CG].
- [3] Wikipedia. *Ray marching*. URL: https://en.wikipedia.org/wiki/Ray_marching (besucht am 2023-08-23).
- [4] Ryan Frazier. *Rendering Constructive Solid Geometry With Python*. 2021. URL: <https://www.fotonixx.com/posts/efficient-csg/> (besucht am 2023-08-23).
- [5] Inigo Quilez. *Distance Functions*. URL: <https://iquilezles.org/articles/distfunctions/> (besucht am 2023-08-22).
- [6] Teadrinker. *Visualization of SDF ray marching algorithm*. 2021. URL: https://commons.wikimedia.org/wiki/File:Visualization_of_SDF_ray_marching_algorithm.png (besucht am 2023-08-23).
- [7] Alan Zucconi. *Volumetric Rendering: Ambient Occlusion*. 2016. URL: <https://www.alanzucconi.com/2016/07/01/ambient-occlusion/> (besucht am 2023-08-22).
- [8] Inigo Quilez. *Soft Shadows in Raymarched SDFs*. 2010. URL: <https://iquilezles.org/articles/rmshadows/> (besucht am 2023-08-22).