LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 148 – Abgabe zu Aufgabe A214
Wintersemester 2022/23

Valentin Böhm Charalampos Piotopoulos

1 Einleitung

Die Peano-Kurve ist eine raumfüllende Kurve, die 1980 von dem italienischen Mathematiker Giuseppe Peano (1858 - 1932) erfunden wurde. Sie gilt als der Urahn der raumfüllenden Kurven. [5] Raumfüllende Kurven sind Kurven, die eine Fläche (oder im Mehrdimensionalen einen Raum) durchlaufen und dabei jeden Punkt einmal besuchen. Sie weisen in der Regel eine rekursive Struktur auf, wodurch die Auflösung (also die Anzahl der Punkte in der Fläche) beliebig erhöht oder verringert werden kann. Andere Beispiele für raumfüllende Kurven neben der Peano-Kurve sind die Hilbert-Kurve und die Lebesgue-Kurve.[3]
Anwendungen lassen sich unter anderem im Bereich der Datenstrukturen oder in der Bildverarbeitung finden[5], aber auch in der digitalen Kartographie werden raumfüllende Kurven genutzt. Das Grundproblem ist dabei, dass die (idealisierte) Erdoberfläche eine zweidimensionale Fläche darstellt, der Speicher in gängigen Rechnerarchitekturen aber eine lineare, eindimensionale Struktur aufweist. Raumfüllende Kurven stellen eine gute Möglichkeit dar, die zweidimensionale Erdoberfläche auf den eindimensionalen Speicher abzubilden. Dabei wird die Erde - wie in Abbildung 1 dargestellt - in einen Würfel gesetzt, bei dem jede Seite mit einer raumfüllenden Kurve belegt ist. In diesem

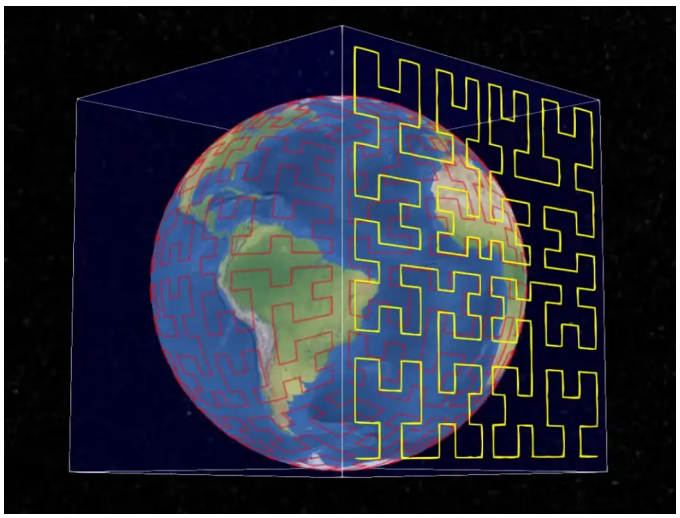

Abbildung 1: Projektion einer raumfüllenden Kurve auf die Erdoberfläche.

Beispiel handelt es sich dabei um eine Hilbert-Kurve, das Prinzip könnte aber genauso mit Peano-Kurven umgesetzt werden. Nun werden diese Kurven auf die Erdoberfläche projiziert. Dabei entsteht eine Kurve auf der Erdoberfläche, wodurch jeder Punkt auf der Erde durch den ihm am nächsten liegenden Punkt der Kurve spezifiziert werden kann. Aufgrund der rekursiven Natur der Kurven kann die Genauigkeit dabei beliebig erhöht oder verringert werden, je nach dem wieviel Rechenleistung und Speicher zur Verfügung steht.[4] Für die Performanz dieser Abbildung ist auch wichtig, dass Punkte, die nah beeinander liegen, auch im Speicher - also auf der Kurve - möglichst nah beeinander liegen. Auch das kann durch raumfüllende Kurven gewährleistet werden. [2]

Wie es für raumfüllende Kurven üblich ist, wird auch die Peano-Kurve nach einer rekursiven Vorschrift gebildet, wobei sie für Grad $N \in \mathbb{N}$ eine Quadrat mit Seitenlänge 3^N bildet. Der exakte Kurvenverlauf ist dabei nicht eindeutig definiert, weil verschiedenen Grundformen zur Auswahl stehen.[5] In dieser Projektaufgabe wurde die in Abbildung 2 dargestellte Grundform verwendet, alle implementierten Algorithmen ließen sich aber ohne großen Aufwand an eine Spiegelung oder Drehung dieser Grundform anpassen.
Um nun eine Peano-Kurve von Grad N zu bilden, wird die Grundform - die gleichzeitig die Kurve für Grad 1 ist und somit ein Gitter der Größe $3 * 3$ auspannt - $(N - 1)$-mal „verfeinert“. Dabei werden jedesmal alle Gitterfelder in neun kleinere Felder zerlegt und diese wiederum mit der Grundform der Kurve belegt, die aber ggf. gespiegelt und/oder gedreht werden muss um sich nahtlos an die Nachbarfelder anzuschließen. Abbildung 2 zeigt die ersten drei Iterationen der Peano-Kurve.

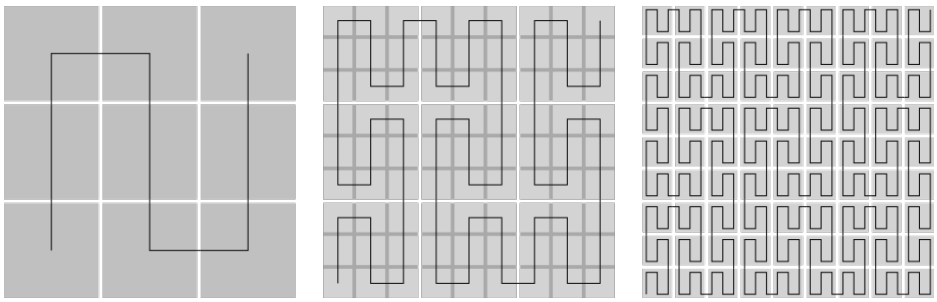

Abbildung 2: Peano-Kurve für n = 1, n = 2 und n = 3

Ziel dieser Projektaufgabe war es, einen Algorithmus für die Generierung einer Peano-Kurve mit gegebenem Grad N zu entwickeln und diesen bzgl. Performanz und Optimierungspotential zu analysieren.

2 Lösungsansatz

Für die Lösung dieser Projektaufgabe wurden drei verschiedene Algorithmen implementiert: Ein iterativer Algorithmus ohne Optimierungen (aufrufbar mit -V0), ein interativer mit Optimierungen (-V1) und - als Vergleich - eine rekursive Variante (-V2).
Der rekursive Algorithmus basiert auf einem 1998 von Greg Breinholt und Christoph Schierz veröffentlichten Algorithmus, der eine mäandernde Peano-Kurve, also eine Peano-Kurve mit sich ändernder Grundform generiert. [1] Dieser wurde an die hier verwendete, nicht mäandernde Kurve angepasst. Die Grundidee ist, Felder so lange in neun kleinere Teilfelder zu zerlegen, bis die Seitenlänge 1 erreicht wird. Für jedes Teilfeld findet ein rekursiver Aufruf statt, dem jeweils Informationen über die Position des Teilfelds und die Ausrichtung der Kurve in der nächsthöheren Ebene übergeben werden.
Der iterative Algorithmus verfolgt einen grundlegend anderen Ansatz. Für jeden Punkt der Zielkurve (die Zielkurve besteht aus 3^{2N} Punkten) wird zunächst die Position innerhalb der Grundkurve ($n = 1$), in der der Punkt liegt, bestimmt. Anschließend werden die Koordinaten mittels Verschiebungen und Spiegelungen Stück für Stück in höhere Grade befördert, bis der Zielgrad erreicht ist. Welche Operationen dafür notwendig sind lässt sich mittels einfacher mathematischer Berechnungen direkt aus dem Index des Punktes in der Zielkurve bestimmen. Konkret werden für jeden Index der Zielkurve (also 0 bis einschl. $3^{2N} - 1$) auf folgende Art die Koordinaten bestimmt:

- Bestimme mit $index \mod 9$ die Koordinaten innerhalb der Grundkurve
- Für $n = 2$ bis N: Berechne $(index/9^{n-1}) \mod 9$ und transformiere die Koordinaten abhängig davon in den nächsthöheren Grad

Am besten lässt sich dieser Algorithmus anhand eines Beispiels nachvollziehen: Sei $N = 3$ und $index = 37$, es sollen also die Koordinaten des 37. Punktes der Peano-Kurve vom Grad 3 bestimmt werden. Zunächst wird wie oben beschrieben die Postition innerhalb der Grundkurve bestimmt. Es gilt $index \mod 9 = 1$, der Punkt hat innerhalb der Kurve vom Grad 1 also den Index 1 und somit die Koordinaten $(x = 0, y = 1)$. Nun werden diese Koordinaten in die höheren Grade, zunächst also in Grad 2 transformiert. Dazu wird $(index/9^{n-1}) \mod 9 = (index/9^1) \mod 9 = 4$ berechnet. Die Kurve vom Grad 1, in der sich der gesuchte Punkt befindet, hat innerhalb der Grad-2-Kurve also den Index 4, was eine Spiegelung in x- und y-Richtung sowie eine Verschiebung um 3^{n-1} in beide Richtungen bedeutet. Die resultierenden Koordinaten lauten $(x = 5, y = 4)$. Zuletzt müssen diese Koordinaten noch in den Grad 3 befördert werden, da in diesem Fall allerdings $(index/9^{n-1}) \mod 9 = (index/9^2) \mod 9 = 0$ gilt, ist dafür keine weitere Transformation nötig. Der Punkt mit Index 37 hat also in der Grad-3-Kurve die gleichen Koordinaten wie im Grad 2, nämlich $(x = 5, y = 4)$.
Die Symmetrie-Eigenschaften der Peano-Kurve können genutzt werden, um diesen Algorithmus zu optimieren: Die Kurve kann in drei Spalten mit Breite 3^{N-1} aufgeteilt werden. Wenn man diese Spalten betrachtet, lässt sich feststellen, dass die dritte Spalte identisch zur ersten ist und die zweite Spalte lediglich eine Spiegelung der ersten an der

Horizontalen darstellt. Die erste Spalte selbst wiederum kann in drei Zeilen der Höhe 3^{N-1} aufgeteilt werden (also drei Kurven vom Grad $N-1$). Man stellt fest, dass die oberste Teilkurve identisch zur untersten ist und die mittlere Kurve mittels Spiegelung der untersten Kurve an der Vertikalen gebildet werden kann. Es genügt also, die erste Teilkurve (die ersten $3^{2(N-1)}$ Punkte) zu bestimmen, anschließend mittels Verschiebung und Spiegelung ebendieser die erste Spalte zu bilden und daraus schließlich die anderen beiden Spalten zu generieren. Außerdem wurden einige Vergleichsoperationen im ursprünglichen Algorithmus durch einfache arithmetische Operationen ersetzt. Da für die Berechnung der Koordinaten der einzelnen Punkte die vorherigen Zwischenergebnisse nicht benötigt werden, wäre auch eine Optimierung mit SIMD denkbar, wodurch die Koordinaten mehrerer Punkte gleichzeitig bestimmt werden könnten. Darauf wurde im Rahmen dieser Projektarbeit aber verzichtet.
Neben der Entwicklung des Algorithmus musste auch ein geeigneter Datentyp für die Koordinaten gewählt werden. Dafür wurde der Typ *unsigned long int* gewählt, da dieser eine Größe von mindestens 32 bit garantiert. Somit können Peano-Kurven bis zu einer Seitenlänge von 2^{32} konstruiert werden, der maximale Grad ergibt sich also aus $3^N \leq 2^{32}$ und es folgt $N \leq 20$. Mit einem Datentyp der Größe 64 bit könnten theoretisch Kurven bis einschließlich Grad 40 generiert werden, dies stellt in der Praxis jedoch keine Verbesserung dar, da das tatsächliche Limit deutlich unter 20 liegt: Um die Koordinaten der Punkte der Kurve vom Grad N als 32 bit Zahlen zu speichern wird eine Speicherkapazität von $3^{2N} * 8$ Byte benötigt, was für Grad 20 einen hypothetischen Speicherbedarf von ca. 100 Exabyte bedeuten würde. Aus diesem Grund wäre eine Vergrößerung des Koordinaten-Datentyps nicht zielführend. Eine Verkleinerung des Datentyps auf 16 bit würde zwar den Speicheraufwand reduzieren, aber auch den maximal berechenbaren Grad auf 10 verringern. Da höhere Grade als 10 für Hochleistungsrechner durchaus machbar sind (für Grad 11 werden etwa 250GB Speicher benötigt), wäre das eine deuliche Einschränkung.

3 Korrektheit

Der iterative Algorithmus basiert in erster Linie auf der Konvertierung von Index zu Koordinaten, für einen bestimmten Punkt auf der gesuchten Kurve können also die Koordinaten bestimmt werden, wenn der Index des Punktes bekannt ist. Um die Korrektheit dieser Konvertierung nachzuvollziehen, ist es hilfreich sie in zwei Teile zu teilen: Im ersten Teil wird die Position des Punktes innerhalb der Kurve vom Grad 1 bestimmt. Da diese Kurve aus genau neun Punkten besteht, befindet sich der gesuchte Punkt an der Stelle $index \mod 9$ in der Kurve. Anhand dieser Information können die Koordinaten - die für die Kurve vom Grad 1 ja bekannt und eindeutig sind - einfach abgelesen werden.

Im zweiten Teil werden die gefundenen Koordinaten nach und nach in höhere Grade transformiert. Dazu wird jeweils die Position der Kurve des aktuellen Grades in der sich der gesuchte Punkt befindet innerhalb der Kurve des nächsthöheren Grades bestimmt. Um die Koordinaten aus Grad n-1 in Grad n zu transformieren wird $(index/9^{n-1})$

$\mod 9$ berechnet. 9^{n-1} ist dabei die Anzahl der Punkte in der Kurve des niedrigeren Grades, durch die Ganzzahldivision $index/9^{n-1}$ wird also bestimmt, an welchem Index sich die (n-1)-Kurve innerhalb der n-Kurve befindet. Die Modulo-Operation wird anschließend benötigt, weil jede Kurve vom Grad n aus neun Kurven mit Grad n-1 besteht.

Aus dem gefundenen Index kann nun mithilfe der Symmetrie-Eigenschaften der Peano-Kurve abgeleitet werden, wie die Koordinaten transformiert werden müssen. Dabei gibt es zwei Arten von Transformation: Verschiebung und Spiegelung.
Die nötigen Verschiebungen lassen sich direkt aus der Grundkurve ablesen und sind in Abbildung 3 dargestellt, wobei L die Seitenlänge der Kurve des niedrigeren Grades darstellt, also 3^{n-1}. Wenn Koordinaten beispielsweise in den Index 7 des nächsthöheren Rang transformiert werden sollen, ist eine Verschiebung um 2L in x-Richtung und L in y-Richtung notwendig.
Die nötigen Spiegelung lassen sich aus der Grundkurve nicht direkt ablesen, hier ist es

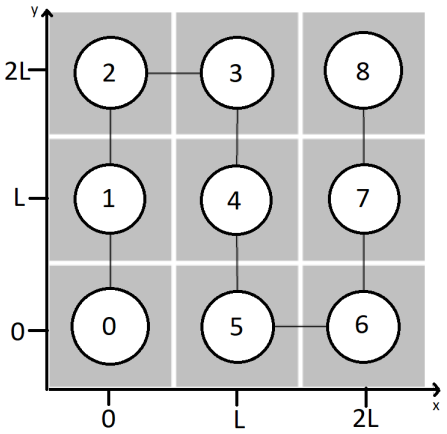

Abbildung 3: Indizes der Peanokurve und jeweils nötige Verschiebungen.

hilfreich eine Kurve höheren Grades, bspw. $n = 2$, zu betrachten. Es lässt sich feststellen, dass die Teilkurven an den Postionen 1, 4, und 7 im Vergleich zur Grundkurve an der Vertikalen und die Teilkurven an den Positionen 3, 4 und 5 an der Horizontalen gespiegelt sind. Um Koordinaten an der Vertikalen zu spiegeln, wird $x = L/2 + (L/2 - x)$ berechnet, wobei wie oben $L = 3^{n-1}$ gilt und die Divisionen Ganzzahldivisionen darstellen. Die Spiegelung an der Horizontalen funktioniert analog mit der y-Koordinate. Diese Transformationen werden nur wiederholt, bis der Zielgrad erreicht ist.

Auch die Korrektheit der Optimierungen lässt sich leicht nachvollziehen. Die oben bereits beschriebenen Symmetrie-Eigenschaften können - wie im Kapitel „Lösungsansatz“ dargelegt - genutzt werden, um die Berechnung der ganzen Kurve auf eine Teilkurve zu reduzieren. Außerdem können teilweise Vergleiche durch einfache arithmetische Operationen ersetzt werden, wenn es um die Bestimmung der nötigen Verschiebungen geht. Für $z \in \mathbb{N}, z < 9$ gilt trivialerweise $z \in \{1, 4, 7\} \iff z \mod 3 = 1$ und

$z \in \{3, 4, 5\} \iff z/3 = 1$, wobei die Division auch hier eine Ganzzahldivision darstellt.

Zu beachtende Sonderfälle in der Implementierung existieren vorallem in Bezug auf den Speicheraufwand, da dieser schon für kleine Grade enorm groß wird. Aus diesem Grund erlaubt das Programm nur Eingaben von $n \leq 20$. Außerdem wird das Programm mit einer Fehlermeldung beendet, wenn nicht genug Speicher alloziert werden kann. Auf normalen Heimrechnern ist dies in der Regel spätestens für $n = 11$ der Fall. Auch fehlerhafte Nutzereingaben (bspw. nicht-positive Grade) werden abgefangen und führen zu einer kontrollierten Beendigung des Programms.
Die exponentiell steigende Anzahl an Punkten führt auch dazu, dass Datentypen mit großem Bedacht gewählt werden müssen. Variablen, die in der Lage sein müssen über alle Punkte zu iterieren benötigen einen Wertebereich der mindestens die Größe 3^{40} hat und brauchen daher Datentypen, die eine Länge von mindestens 64 bit garantieren, denn $2^{64} > 3^{40} > 2^{32}$. Dafür wurde der Typ *unsigned long long int* gewählt.
Abbildung 4 zeigt die Ausgabe des Programms für die Grade 1, 3 und 5.

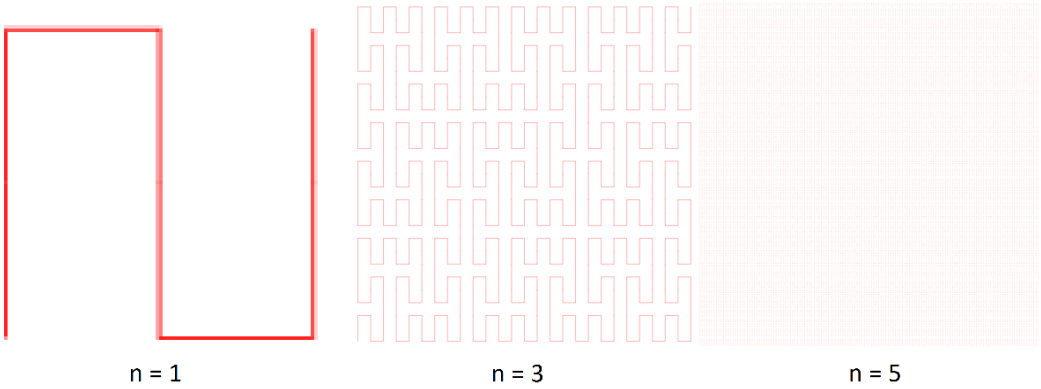

Abbildung 4: Programmausgabe für die Grade 1, 3 und 5.

4 Performanzanalyse

Um die drei Algorithmen (Iterativ mit Optimierungen, Iterativ ohne Optimierungen, Rekursiv) bzgl. ihrer Laufzeit zu vergleichen, wurden sie mit den Eingaben 2 bis 9 ausgeführt und jeweils die durchschnittliche Laufzeit über zehn Iterationen bestimmt. Getestet wurde auf einem Ubuntu-System mit AMD Ryzen 7 5700u (8 Kerne, 1.8 GHz) und 16 GB Arbeitsspeicher. Auf Messungen mit höheren Graden wurde verzichtet, da für Grad 10 bereits eine Speicherkapazität von mindestens $3^{20} * 8$ Byte benötigt wird, was ca. 28 GB entspricht und daher auf dem Testgerät nicht möglich war. Für die Testläufe wurde die Compileroption -O2 verwendet. Die Ergebnisse sind in Abbildung 5 dargestellt.
Es ist klar zu erkennen, dass alle drei Algorithmen eine exponentielle Laufzeit aufweisen, was nicht weiter verwunderlich ist, da die Zahl der Punkte exponentiell steigt. Auf den ersten Blick mag es überraschend sein, dass die Laufzeit des rekursiven Algo-

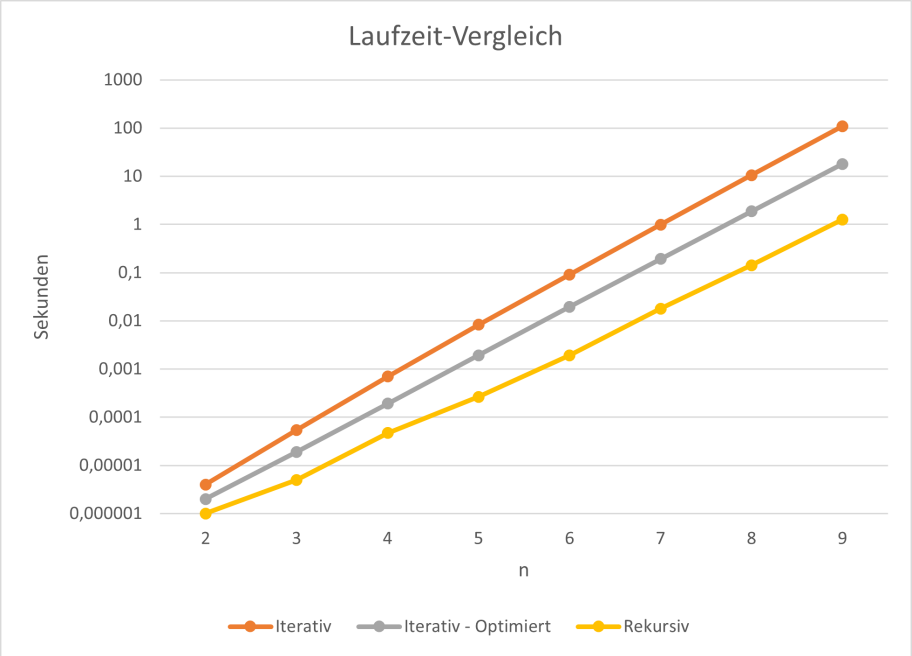

Abbildung 5: Laufzeit der verschiedenen Algorithen in Abhängigkeit des Grades n

rithmus - insbesondere für große n - deutlich unter der der iterativen Varianten liegt. Dies lässt sich jedoch nachvollziehen, wenn man die asymptotische Laufzeit der Algorithmen berechnet und vergleicht: Der iterative Algorithmus (ohne Optimierung) führt für jeden der 3^{2n} Punkte n Schleifendurchläufe aus. Die asymptotische Laufzeit liegt also in $O(3^{2n} * n)$. Durch die Optimierungen wird die Laufzeit um den kostanten Faktor 9 verbessert, da nur eine der neun Teilkurven berechnet werden muss. An der asymptotischen Laufzeit ändert das nichts, da konstante Faktoren dafür nicht relevant sind. Die tatsächliche Laufzeit für n im „realistischen“ Bereich profitiert davon aber ganz erheblich, beispielsweise beträgt die Laufzeit des nicht optimierten Algorithmus in der beschriebenen Testumgebung für $n = 9$ etwa 110 Sekunden, die des optimierten Algorithmus nur rund 18 Sekunden.
Der rekursive Algorithmus führt für Grad n genau $1 + 9 + 9^2 + ... + 9^n$ Funktionsaufrufe durch, die asymptotische Laufzeit liegt also in $O(9^n)$ bzw. $O(3^{2n})$ und ist somit um dem Faktor n besser als die der iterativen Variante. In der Praxis - also für n im „realistischen“ Bereich - ist die Verbesserung sogar noch deutlich höher, weil die rekursive Variante deutlich weniger Vergleichsoperationen und arithmetische Berechnungen durchführen muss. So beträgt die Laufzeit für $n = 9$ etwa 1,3 Sekunden und ist damit um weit mehr als den Faktor n schneller als die iterativen Algorithmen.

5 Zusammenfassung und Ausblick

In dieser Projektaufgabe wurden zunächst die Grundlagen der Peano-Kurve und ein konkretes Anwendungsbeispiel erläutert. Anschließend wurden drei Algorithmen zur Generierung von Peano-Kurven implementiert, vorgestellt und analysiert. Der Fokus lag dabei auf einem iterativen Algorithmus, der die Symmetrie-Eigenschaften der Peano-Kurve nutzt um mit möglichst wenig Rechenaufwand auszukommen. Als Vergleich diente eine rekursive Variante, die sich - sowohl in der Theorie als auch in der Praxis - als performanter herausstellte. Beide Ansätze werden jedoch durch ihren massiven Speicherverbrauch limitiert. Dieser lässt sich nicht ohne Einschränkungen optimieren, da er direkt durch die Anzahl der Punkte bestimmt wird.
Die Laufzeit des iterativen Alogrithmus könnte potentiell durch die Nutzung von SIMD-Instruktionen verbessert werden. Dabei würden mehrere Punkte gleichzeitig berechnet werden, es wäre also eine Verbesserung um einen kleinen konstanten Faktor zu erwarten. Die asymptotische Laufzeit der rekursiven Variante könnte dadurch aber nicht übertroffen werden.
Abschließend lässt sich festhalten, dass die Berechnung von großen Peano-Kurven mit hohem Speicherverbauch und Rechenaufwand verbunden ist. Um Kurven vom Grad 10 oder höher effizient berechnen zu können, wären Hochleistungsrechner oder grundlegend andere konzeptionelle Ansätze notwendig.

Literatur

[1] Greg Breinholt and Christoph Schierz. A recursive algorithm for the generation of space-filling curves. 1998.

[2] David Cavender. Space-filling curves & generalizing the peano curve. 2019. `https://doi.org/10.17615/959t-tr73`.

[3] Hans Sagan. *Space-Filling Curves*. Springer-Verlag Berlin Heidelberg New York, 1994.

[4] Sven Kreiss. S2 cells and space-filling curves: Keys to building better digital map tools for cities. `https://medium.com/sidewalk-talk/s2-cells-and-space-filling-curves-keys-to-building-better-digital-map-tools-for-cities-a312aa5e2f59`, 2016. Letzter Zugriff am 19. Januar 2023.

[5] Michael Bader. *Space-Filling Curves - An Introduction with Applications in Scientific Computing*. Springer-Verlag Berlin Heidelberg, 2013.