Institut für Informatik
der Technischen Universität München

Lehrstuhl für numerische Programmierung
und Ingenieuranwendungen in der Informatik

Raumfüllende Kurven

Begleitendes Skriptum
zum entsprechenden Kapitel der Vorlesung

„Algorithmen des Wissenschaftlichen Rechnens“

– gehalten im Sommersemester 2004 –

Dr. Michael Bader

Dieses Skript ist eine vorläufige Fassung und enthält vermutlich noch einige Fehler. Bitte stets mit den Vorlesungsfolien abgleichen und mögliche Fehler oder Unklarheiten an Michael Bader (`bader@in.tum.de`) melden. Dies gilt in gleicher Weise für alle Verbesserungsvorschläge.

– Fassung vom 5. Juli 2006 –

Inhaltsverzeichnis

1 Linearisierung multidimensionaler Daten

Als multidimensionale Daten werden wir in diesem Skript alle Datensätze ansehen, bei denen einem n-Tupel von Indizes, also einem $(i_1, i_2, \ldots, i_n)$ jeweils ein Datum zugeordnet ist. In praktisch allen Programmiersprachen werden mehrdimensionale Felder als Datenstrukturen für derartige Daten zur Verfügung gestellt. Interessanterweise ist die Implementierung dieser Datenstrukturen von Programmiersprache zu Programmiersprache durchaus verschieden. Während etwa Pascal und FORTRAN eine zeilen- bzw. spaltenweise Aufzählung der Elemente bevorzugen, verwenden C/C++ und Java einen verschachtelte Zeigerstruktur.

Beispiele für multidimensionale Daten
Beispiele für multidimensionale Daten gibt es fast so viele, wie Anwendungen in der der Informatik. Wir wollen folgende, besonders wichtige herausgreifen:

- Vektoren, Matrizen, Tensoren, etc. in der linearen Algebra
- alle Arten von (Raster-)Bilddaten, zum Beispiel digitalisierte Bilder, Tomographie-Daten, Filme und Animationen (bewegte Bilder)
- die Diskretisierung physikalischer Modelle auf (regelmäßigen) Diskretisierungsgittern führen zu multidimensionalen Daten – sie treten somit bei der Lösung von (partiellen) Differentialgleichungen auf. Nicht nur die Gitterdaten, sondern vor allem die Rechenergebnisse sind multidimensionale Daten.
- Koordinaten aller Art (oft in Verbindung mit Graphen)
- Tabellen, diese vor allem als wesentliche Datenstruktur relationaler Datenbanken

- viele statistische Daten sind multidimensional; zum Beispiel werden in der Finanzmathematik oft „Körbe“ aus unterschiedlichen Aktien oder Optionen betrachtet.

Beispiele für Algorithmen/Operationen
Als typische Operationen, die wir auf diesen Datenstrukturen durchführen, können wir folgende identifizieren:

- in der linearen Algebra die klassischen Matrizenoperationen (Matrixmultiplikation, Lösen von Gleichungssystemen, etc.)
- Durchlaufen der Daten (Traversierung); z.B. wenn jedes Datum auf eine vorgegebene Weise modifiziert oder aktualisiert werden muss
- das Speichern und Laden multidimensionaler Datensätze, sei es im Arbeitsspeicher oder in einer Datei
- das Auffinden eines bestimmten Datums bzw. das Selektieren eines Teildatensatzes (z.B. eines bestimmten Bildausschnitts)
- die Partitionierung der Daten, d.h. ihre Zerlegung in gleich große Teile. Dies ist vor allem bei der Parallelisierung oder bei divide-and-conquer-Algorithmen von praktischer Relevanz.
- der Zugriff auf benachbarte Elemente (z.B. die Diskretisierung partieller Differentialgleichung liefert häufig Beziehungen zwischen direkt benachbarten Gitterpunkten).

In mehreren dieser Operationen können wir als besonders wichtiges Teilproblem die *Linearisierung* ausmachen:

- das sequentielle Durchlaufen der Daten ist ein solcher Sonderfall der Linearisierung;
- ebenso das Speichern von Daten im Hauptspeicher oder auf der Festplatte, da dies zwangsläufig stets in sequentieller Reihenfolge geschieht.
- das Linearisieren von Daten schafft zudem eine Ordnung, nach der die Daten sortiert werden können – dadurch werden oft nachfolgende Operationen vereinfacht.

Ganz allgemein können wir jede Operation, die aus verschachtelten Schleifen, also Konstrukten wie

```
for i from 1 to n do
    for j from 1 to m do
        ...
```

als Operation oder Traversierung auf multidimensionalen Daten auffassen.

1.1 Anforderungen an die Linearisierung

Nachdem wir die Linearisierung als wichtiges Problem erkannt haben, wollen wir untersuchen, welche Anforderungen an die Linearisierung zu stellen sind. Wann ist eine Linearisierung gut und wann ist sie schlecht? Dabei sind einige Forderungen zwingende Voraussetzung, andere wiederum sind lediglich für bestimmte Anwedungsfälle besonders günstig.

Zwingend notwendige Eigenschaften der Linearisierung:

- Eine zwingende Forderung an die Linearisierung ist, dass sie eine eindeutige Nummerierung der Daten erzeugt – jedem Datum sei ein eindeutiger *Index* zugeordnet. Nur dies stellt sicher, dass z.B. beim Speichern auf Festplatte und anschließendem Laden der Daten wieder die selben Daten im Arbeitsspeicher vorliegen. Ebenso muss bei der Traversierung sichergestellt sein, dass jedes Datum genau einmal behandelt wird. Wenn wir die Zuordnung zwischen Nummern und Daten als mathematische Abbildung auffassen, entspricht diese Eineindeutigkeit gerade der Forderung, dass die Abbildung *bijektiv* sein muss.

- Zu einem zusammenhängenden Datensatz soll auch die Linearisierung eine durchgehende Folge von Indizes sein – idealerweise von 0 bis $n-1$ (je nach Geschmack auch von 1 bis n). „Löcher“ im Indexbereich sollen nicht auftreten. Natürlich lässt sich darüber streiten, ob dies nicht bereits eine Forderung bzgl. der Effizienz ist – ein kleiner Overhead an unbenutzten Nummern wäre in der Praxis oft tolerierbar.

Anforderungen bzgl. der Effizienz:

Die folgenden Anforderungen ergeben sich allein aus den jeweiligen Anwendungen. In unterschiedlichen Anwendungen werden bisweilen nur einige wenige überhaupt eine Rolle spielen. Und in praktisch allen Anwendungen werden die Anforderungen unterschiedliche Bedeutung haben.

- Als vielleicht naheliegendste Forderung wollen wir festhalten, dass die Abbildung zur Linearisierung einfach und schnell zu berechnen sein soll. Als Mindestkriterium muss sie so schnell berechenbar sein, dass sie anderweitig erzielte Vorteile nicht wieder zunichte macht.

- Die Linearisierung soll Nachbarschaftsbeziehungen erhalten. Wenn zwei Daten im multidimensionalen Raum nahe beieinander liegen, soll dies auch in der Linearisierung so bleiben. Diese Forderung ist immer dann wichtig, wenn auf benachbarte Daten oft gleichzeitig (z.B. extrahieren von Bildausschnitten) oder kurz aufeinander folgend zugegriffen wird. Die Nachbarschaftserhaltung erzeugt dann eine *Datenlokalität*, die sich z.B. günstig auf die Ausnutzung von Caches auswirkt.

- Zwei Ausprägungen dieser Nachbarschaftserhaltung sind die *Stetigkeit* der Linearisierungsabbildung und die *Clusterung* der Daten. Die Stetigkeit bewirkt, dass Daten mit aufeinander folgendem Index im mehrdimensionalen Datenraum stets direkt benachbart sind. Stark geclusterte Daten liegen dann vor, wenn der zu einem bestimmten Indexbereich gehörige Datenbereich in keiner Richtung stark ausgedehnt ist. Im 3D würde man sich möglichst kugelförmige Bereiche wünschen.

- Die Linearisierung soll keine einzelnen Dimensionen bevorzugen oder benachteiligen. Diese Forderung mag zunächst einem allgemeinen Gerechtigkeitsgefühl entstammen. Wenn jedoch die Vorhersagbarkeit der Laufzeit von Programmen zu einem Kriterium wird, dann gewinnt diese Eigenschaft an Bedeutung.

1.2 Arten der Linearisierung

Die bei weitem gebräuchlichste Art der Linearisierung multidimensionaler Daten ist das *zeilenweise* oder *spaltenweise* Aufzählen der Daten. Für ein digitalisiertes Bild werden zum Beispiel alle Pixel von links nach rechts und zeilenweise von oben nach unten aufgeführt. Bei der Speicherung von Matrizen ist in vielen Programmiersprachen die zeilenweise (*row-major*) oder spaltenweise (*column-major*) Durchnummerierung üblich. Beim Zugriff auf die Elemente wird dann meist mit den folgenden Operationen zwischen Indexpaar und Adresse im Speicher umgerechnet:

$$\text{address}(A_{ij}) = in + j \qquad \text{oder} \qquad \text{address}(A_{ij}) = i + jm,$$

n sei dabei die Zahl der Spalten und m die der Zeilen der Matrix. Die zeilenweise Nummerierung (linke Formel) wird zum Beispiel in den Programmiersprachen Pascal und C (dort nur für arrays mit festen Dimensionsgrößen) verwendet. Die spaltenweise Nummerierung wird von Fortran verwendet.

Welche Anforderungen erfüllt die zeilen- bzw. spaltenweise Aufzählung?
Die im vorigen Artikel formulierten Anforderungen erfüllen die zeilen- oder spaltenweise Aufzählung nur teilweise:

- Die Linearisierung ist zunächst leicht und schnell zu berechnen.
- Die Erhaltung von Nachbarschaftsbeziehung funktioniert nur in einer der beiden Raumrichtungen. Bei der zeilenweisen Nummerierung etwa sind jeweils die links und rechts benachbarten Elemente auch im Speicher in benachbarten Speicherzellen abgelegt. Nach oben bzw. unten benachbarte Elemente sind aber mindestens soweit entfernt, wie die Matrix Spalten hat.

  Bei höherdimensionalen Datenstrukturen wird diese Erhaltung sogar noch schlechter – es bleibt immer nur die Nachbarschaft in einer Raumrichtung erhalten.
- Die Stetigkeit ist jeweils an den Rändern der Matrix verletzt – zwischen zwei im Speicher benachbarten Elementen kann hier ein Zeilensprung auftreten. Die Elemente liegen dann an entgegen gesetzten Rändern der Matrix.

Eine Clusterung von Daten liegt überhaupt nicht vor. Eine im Speicher direkt aufeinander folgende Sequenz von Elementen liefert im Allgemeinen ein schmales Band in der Matrix. Dieses Band ist stets so breit wie die Matrix, außer wenn der Indexbereich kleiner ist als die Anzahl der Spalten. In diesem Fall ist das Band aber nur maximal ein Element hoch.

- Von einer Gleichbehandlung der Dimensionen kann ebenfalls keine Rede sein. Zum Beispiel findet man in Büchern über effiziente Programmierung oft den Rat, dass über zeilenweise nummerierte Spalten alle Schleifen nach Möglichkeit so programmiert werden sollen, dass die Elemente auch zeilenweise abgearbeitet werden. Die zur Verfügung stehenden Compiler und Prozessoren sind so gebaut, dass ein solches Programm deutlich schneller ausgeführt wird, als wenn die Schleife entgegen der Nummerierungsrichtung ausgeführt wird.

Der letzte Punkte war ein Beispiel dafür, dass die Forderungen gerade in Hinblick auf die Effizienz der Implementierung gestellt wurden.

Effizienz der zeilen- bzw. spaltenweisen Aufzählung?
Dass die zeilen- oder spaltenweise Nummerierung von Matrixelementen eher ungünstig ist, sieht man vor allem an dem Aufwand, der in Lineare-Algebra-Bibliotheken getrieben wird, um die typischen Matrixoperationen schnell zu implementieren. Hier zeigt sich, dass die zeilen- oder spaltenweise Nummerierung für die Effizienz oft kontraproduktiv ist. Fast immer wird durch Aufteilen der Matrixoperationen in Operationen auf kleineren Matrixblöcken versucht, die Datenlokalität zu erhöhen. Denn nur dann werden die verhältnismäßig kleinen Rechnercaches gut ausgenutzt.

Wegen ihrer Einfachheit ist die zeilen- oder spaltenweise Nummerierung trotzdem so fest in den Köpfen der meisten Programmierer „verdrahtet“, dass eigentlich kaum noch über mögliche Alternativen nachgedacht wird. Die naheliegendste Wahl der Schleife, ein

```
for i from 1 to n do
    for j from 1 to m do
        ...
```

bleibt dadurch fast immer in der endgültigen Fassung eines Programms erhalten. Oft beginnt man erst dann über Verbesserungen nachzudenken,

wenn sich zeigt, dass die maximale Rechenleistung eines Prozessors nur zu Bruchteilen ausgenutzt wird, weil der Prozessor im Wesentlichen damit beschäftigt ist, auf den langsamen Hauptspeicher zu warten. Die Wirksamkeit des Caches wird durch die schlecht gewählte Datenstruktur ausgehebelt.

Wir werden uns daher in diesem Skript mit einer anderen Möglichkeit beschäftigen, multidimensionale Daten zu linearisieren – mit den sogenannten *raumfüllenden Kurven*. Die raumfüllenden Kurven sind auch ein Beispiel dafür, dass Konstrukte und Ideen, die anfangs nur in sehr theoretischen Arbeiten und Überlegungen beheimatet waren, auf einmal eine höchst spannende Anwendung in stark praxisorientierten Arbeitsgebieten finden.

2 Raumfüllende Kurven

2.1 Abbildungen zwischen Intervallen und Flächen

Die *Mächtigkeit* von Mengen, also die Charakterisierung der Anzahl ihrer Elemente, ist sowohl in der Mathematik als auch in der Informatik ein wichtiger Begriff. In der Informatik, zum Beispiel, basieren darauf die Begriffe der Aufzählbarkeit und Abzählbarkeit, aus denen sich viele Ergebnisse zur Berechenbarkeit von Funktionen ergeben.

Als kontraintuitives Beispiel zur Mächtigkeit von Mengen findet man in vielen Büchern und Vorlesungen die folgende Abildung zwischen dem Einheitsintervall $[0,1]$ und dem Einheitsquadrat $[0,1] \times [0,1]$. Dieser Abbildung liegt eine recht einfache Idee zugrunde, sofern man die Binärdarstellung von Zahlen verwendet. Zu einem Argument $t \in [0,1]$ berechne man also dessen Binärdarstellung

$$t = 0_2.b_1b_2b_3b_4b_5b_6\ldots$$

Als Bild von t unter der zu beschreibenden Abbildung f legt man dann den Punkt

$$f(t) = \begin{pmatrix} 0_2.b_1b_3b_5\ldots \\ 0_2.b_2b_4b_6\ldots \end{pmatrix}$$

fest. Man bildet also die erste Komponente aus den „ungeraden“ Binärstellen und die zweite Komponente aus den „geraden“ Stellen. Umgekehrt lässt sich analog eine Abbildung g vom Einheitsquadrat auf das Einheitsintervall definieren:

$$g\begin{pmatrix} 0_2.b_1b_3b_5\ldots \\ 0_2.b_2b_4b_6\ldots \end{pmatrix} = 0_2.b_1b_2b_3b_4b_5b_6\ldots$$

Man sieht leicht, dass sowohl f als auch g *surjektive* Abbildungen sind – jeder Punkt des Einheitsquadrats tritt als Bild von f und jede Zahl im

Einheitsintervall als Ergebnis von g auf. Wären Einheitsquadrat und Einheitsintervall endliche Mengen, könnten wir daraus folgern, dass sie gleich mächtig sein müssen. Wie aber verhält es sich für unendliche Mengen?

Zwei endliche Mengen werden genau dann als gleich mächtig bezeichnet, wenn es eine *bijektive* Abbildung zwischen den beiden Mengen gibt. Diese Interpretation hat sich auch für unendliche Mengen als geeignet erwiesen. Weder f noch g sind jedoch bijektiv. Zum Beispiel gilt:

$$f\left(\frac{1}{2}\right) = f(0_2.1) = \begin{pmatrix} 0_2.1 \\ 0_2.0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix}$$

und

$$f\left(\frac{1}{6}\right) = f(0_2.0010101\ldots) = \begin{pmatrix} 0_2.01111\ldots \\ 0_2.00000\ldots \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix}.$$

Bereits im Jahr 1878 zeigte jedoch Georg Cantor die Existenz einer bijektiven Abbildung zwischen Einheitsintervall und Einheitsquadrat (als Korrollar eines allgemeineren Satzes). Damit war auch gezeigt, dass das Einheitsintervall gleich viele Punkte enthält wie das Quadrat – obwohl das Intervall eindeutig eine echte Teilmenge des Einheitsquadrats ist. Dieser scheinbare Widerspruch führte zu anhaltenden Diskussionen über die Mächtigkeit von Mengen. Heute sind in der Informatik daraus resultierende Begriffe wie die *Abzählbarkeit* Teil jeder theoretischen Vorlesung.

Schon kurz nach Cantors Veröffentlichung begannen mehrere Mathematiker nach Abbildungen zu suchen, die das Einheitsintervall nicht nur bijektiv sondern auch stetig auf das Quadrat abbildeten. Der Graph derartiger Abbildungen wird auch eine *Kurve* genannt, so dass man also Kurven suchte, die ein Quadrat komplett ausfüllen (Surjektivität) sich aber nicht selbst schneiden oder berühren (Injektivität). Bereits 1879 wurde diese Suche von Eugen Netto beendet, und zwar negativ: eine derartige Abbildung kann nicht zugleich bijektiv und stetig sein – sofern man nicht anstelle des Quadrats viel unregelmäßigere Zielgebiete zulässt (das Zielgebiet darf keine glatten Ränder haben).

Im Jahr 1890 wurden von Giuseppe Peano und von David Hilbert jedoch Kurven gefunden, die stetig und *surjektiv* sind, aber nicht injektiv. Diese *raumfüllenden Kurven* werden wir in diesem Kapitel vorstellen und näher untersuchen. Weitere raumfüllende Kurven gehen auf Henri Lebesgue (1904) und Waclaw Sierpinski (1912) zurück.

2.2 Was ist eine Raumfüllende Kurve?

Im Folgenden werden wir verschiedene *raumfüllende Kurven* vorstellen – zum Beispiel stetige Abbildungen zwischen Intervall und Einheitsquadrat. Bevor wir damit beginnen wollen wir uns kurz Zeit nehmen zu definieren, was wir genau unter dem Begriff „raumfüllende Kurve“ verstehen wollen.

Dazu müssen wir zunächst festlegen, was eine mathematische *Kurve* ist. Unsere Vorstellung einer Kurve ist vermutlich die einer durchgezogenen Linie (oder eines Bogens oder ...). Mathematisch ergibt sich eine Kurve aus der Abbildung eines Parameterintervalls in eine Fläche oder in ein Volumen.

Definition 2.1 (Kurve)
Sei $f\colon \mathcal{I} \to \mathbb{R}^n$ eine stetige Abbildung von der kompakten Menge $\mathcal{I} \subset \mathbb{R}$ in den $\mathbb{R}^n$. Dann ist das zugehörige Bild $f_*(\mathcal{I})$ der Abbildung eine Kurve*, und die Darstellung $x = f(t), t \in \mathcal{I}$ heißt* Parameterdarstellung *der Kurve.*

Dabei ist das *Bild* einer Abbildung definiert als die Menge aller angenommenen Werte, also $f_*(\mathcal{I}) := \{f(x) \in \mathbb{R}^n \mid x \in \mathcal{I}\}$. Als kompakte Parametermengen $\mathcal{I}$ werden wir im Allgemeinen Intervalle betrachten, z.B. das Einheitsintervall $[0, 1]$. Allerdings können auch kompliziertere kompakte Mengen verwendet werden (Lebesgue-Kurve). Anstelle von $\mathbb{R}^n$ könnte auch jeder andere euklidische Vektorraum (d.h. mit Norm/Skalarprodukt) verwendet werden. Wir werden es aber hier beim $\mathbb{R}^n$ belassen.

Als *raumfüllende* Kurve werden wir nun jede Kurve bezeichnen, die ein gewisses Flächenstück oder ein Volumen komplett ausfüllt:

Definition 2.2 (Raumfüllende Kurve)
Eine zu einer Abbildung $f\colon \mathcal{I} \to \mathbb{R}^n$ gehörige Kurve $f_*(\mathcal{I})$ heißt raumfüllende Kurve*, wenn $f_*(\mathcal{I})$ einen Jordaninhalt (Fläche, Volumen, ...) größer als* 0 *besitzt.*

Wenn die Abbildung $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$ *surjektiv* ist, d.h. wenn jeder Wert in der Teilmenge $\mathcal{Q}$ tatsächlich angenommen wird, dann ist $f_*(\mathcal{I})$ genau dann eine raumfüllende Kurve, wenn die Fläche (bzw. das Volumen) von $\mathcal{Q}$ größer als 0 ist.

Falls $\mathcal{Q}$ ein Gebiet mit glattem Rand ist, so haben wir bereits gehört, dass es keine bijektive Abbildung $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$, geben kann, so dass $f_*(\mathcal{I})$ eine raumfüllende Kurve ist (Beweis: E. Netto, 1879). Also kann f nicht injektiv sein, was bedeutet, dass unter Umständen mehrere Parameter auf

den gleichen Bildpunkt abgebildet werden. Wir könnten also in gewissem Sinne behaupten, dass das Parameterintervall nicht nur gleich viele Punkte enthält wie das Zielgebiet Q, sondern sogar mehr als dieses!

2.3 Die Hilbertkurve

Die Konstruktion und spätere Definition der Hilbertkurve als raumfüllende Kurve beruht im Wesentlichen auf einem rekursiven Verfahren:

1. Das zu füllende quadratische Gebiet wird aufgeteilt in vier kongruente Teilquadrate, die entsprechend die halbe Seitenlänge des Ausgangsquadrats besitzen.
2. Finde für jedes Teilquadrat eine raumfüllende Kurve, die durch Spiegelung und/oder Rotation der ursprünglichen Kurve gebildet wird.
3. Die Spiegelungs- und Rotationsoperatoren sind so zu wählen, dass die vier Teilkurven anschließend stetig zusammengefügt werden können.

Wir müssen also insbesondere festlegen, wie die Spiegelungen und Rotationen durchzuführen sind. Außerdem müssen wir uns überlegen, wie wir die Rekursionsidee in eine Definition überführen können.

2.3.1 Iterationen der Hilbertkurve

Ein geeignetes Mittel, um die Spiegelungs- und Rotationsoperatoren sowie die Verbindung der einzelnen Teilkurven zu visualisieren, sind die sogenannten *Iterationen* der Hilbertkurve. Die Iterationen sind Kurven, die als Streckenzug jeweils die Mittelpunkte der rekursiv gebildeten Teilquadrate verbinden. Dabei werden alle Teilquadrate eines rekursiven Level in der Reihenfolge miteinander verbunden, wie die Teilquadrate auch von der Hilbertkurve durchlaufen werden sollen.

Für jeden rekursiven Level erhalten wir demnach eine Iteration der Hilbertkurve. Wir werden von der *n-ten Iteration der Hilbertkurve* sprechen, wenn diese die Teilquadrate nach der n-ten rekursiven Unterteilung verbindet.

Abbildung 2.1 skizziert, wie diese Iterationen schrittweise zu bilden sind. Es gibt zwei gleichwertige Möglichkeiten, diese Konstruktion aufzufassen:

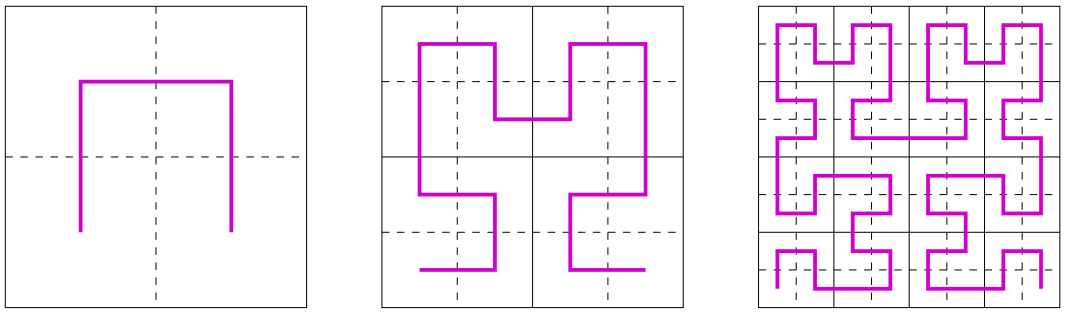


Abbildung 2.1: Die ersten drei Iterationen der Hilbertkurve

1. Bei der Unterteilung eines Teilquadrats in vier kleinere Quadrate wird das Grundmuster (vgl. linkes Bild in Abbildung 2.1) geeignet rotiert und/oder gespiegelt in das Teilquadrat gelegt. Dadurch, dass festgelegt ist, über welche Kanten des ursprünglichen Teilquadrats die Kurve in das Quadrat hinein- und wieder hinausläuft, ist auch bestimmt, wie das Grundmotiv zu rotieren bzw. zu spiegeln ist.

2. Zur jeweils nächsten Iteration werden wir Exemplare der ursprünglichen Iteration geeignet zusammengefügt.

Die Hilbertkurve werden wir uns als Grenzkurve vorstellen, die wir erhalten, wenn wir die rekursive Unterteilung immer weiter fortsetzen. Die Iterationen selbst werden wir jedoch später zur Traversierung z.B. von Datenstrukturen wie Zellgittern oder Matrizen verwenden.

2.3.2 Approximierende Polygone der Hilbertkurve

Ein weiteres Mittel zur Beschreibung der Konstruktion der Hilbertkurve liefern uns die sogenannten *approximierenden Polygone*. Anhand der Iterationen der Hilbertkurve können wir ersehen, dass die „fertige“ Hilbertkurve in der linken unteren Ecke des Quadrats beginnt, das Quadrat komplett durchläuft und in der rechten unteren Ecke endet. Wenn wir die feiner aufgelösten Iterationen betrachten, erkennen wir, dass dies auch für jedes verwendete Teilquadrat gelten wird: die Hilbertkurve beginnt in einem der Ecken des Teilquadrats, durchläuft das entsprechende Teilquadrat und endet in einer Ecke, die an einer gemeinsamen Kante liegt.

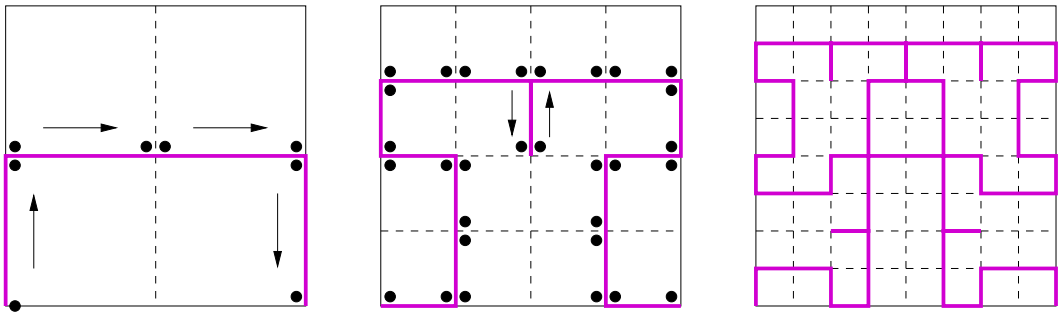


Abbildung 2.2: Die ersten drei approximierenden Polygone der Hilbertkurve; Start- und Endpunkte, sowie die Durchlaufrichtung der Hilbertkurve sind teilweise zusätzlich eingezeichnet.

Die *approximierenden Polygone* bestehen gerade aus den Kanten, die die jeweiligen Start- und Endpunkte in den Teilquadraten verbinden. Wie bei den Iterationen erhalten wir für jede Iterationsstufe ein neues approximierendes Polygons – wir sprechen vom *n-ten approximierenden Polygon*, wenn dieses aus den entsprechenden Kanten der n-ten Verfeinerung besteht.

Abbildung 2.2 zeigt die ersten drei approximierenden Polygone der Hilbertkurve. Dabei sind in den ersten beiden Polygonen zusätzlich die Start- und Endpunkte der Hilbertkurve, sowie teilweise die Durchlaufrichtung für jedes Teilquadrat eingezeichnet. Zusammen mit den Iterationen bilden sie also die Grundlage für die Beschreibung, in welcher Reihenfolge die Teilquadrate von der Hilbertkurve durchlaufen werden. Wir werden in Kapitel 5 noch detaillierter auf die approximierenden Polygone eingehen.

2.3.3 Definition der Hilbertkurve

Mathematisch definieren wir die Hilbertkurve mit Hilfe spezieller Intervallschachtelungen. Grundlage dieser Idee ist, dass die Hilbertkurve, nachdem sie ein Teilquadrat – also ein Viertel des Gesamtquadrats – abgelaufen ist, auch ein Viertel des Parameterintervalls abgelaufen haben soll. Dies entspricht der einleuchtenden Forderung, dass die Hilbertkurve nicht in einigen Bereichen schneller und in anderen Bereichen langsamer voranschreitet.

Wir werden daher einem gegebenen Parameter t eine Intervallschachtelung nach folgendem Muster zuordnen:

- Das erste Intervall ist das gesamte Parameterintervall, also $[0,1]$.
- Das Nachfolgeintervall eines Intervalls $[a, a+h]$ ist stets eines der Intervalle

$$\left[a, a+\frac{h}{4}\right], \left[a+\frac{h}{4}, a+\frac{2h}{4}\right], \left[a+\frac{2h}{4}, a+\frac{3h}{4}\right], \left[a+\frac{3h}{4}, a+h\right],$$

  also das erste, zweite, dritte oder vierte Viertel von $[a, a+h]$.
- Es wird immer dasjenige Viertelintervall ausgewählt, in dem der Parameter t liegt.

Zum Beispiel erhalten wir für $t = \frac{1}{3}$ folgende Intervallschachtelung:

$$[0,1], \left[\frac{1}{4}, \frac{2}{4}\right], \left[\frac{5}{16}, \frac{6}{16}\right], \left[\frac{21}{64}, \frac{22}{64}\right], \ldots$$

Gemäß unserer rekursiven Grundidee wird jedes der Intervalle der Intervallschachtelung auf ein bestimmtes Teilquadrat abgebildet. Wir bekommen also analog zu den Intervallen eine Schachtelung von Quadraten, also zweidimensionalen Intervallen. Das Quadrat $[a,b] \times [c,d]$ hat dabei als Nachfolgequadrat eines der folgenden:

$$\left[a, \frac{a+b}{2}\right] \times \left[c, \frac{c+d}{2}\right], \quad \left[\frac{a+b}{2}, b\right] \times \left[c, \frac{c+d}{2}\right],$$
$$\left[a, \frac{a+b}{2}\right] \times \left[\frac{c+d}{2}, d\right], \quad \left[\frac{a+b}{2}, b\right] \times \left[\frac{c+d}{2}, d\right].$$

Welches Nachfolgequadrat wir konkret wählen müssen, können wir aus den Iterationen oder auch aus den approximierenden Polygonen der Hilbertkurve ablesen. Eine erläuternde Skizze findet sich in Abbildung 2.3.

Falls t gerade auf einer Intervallgrenze liegt, können wir zwei verschiedene Intervallschachtelung (und resultierende Quadratschachtelungen) für dieses t erhalten. Ob daraus Probleme resultieren, werden wir untersuchen, nachdem wir die Hilbertkurve definiert haben.

Definition 2.3 (Hilbertkurve)
Die Parameterdarstellung $h(t)$ sei definiert durch folgende Abbildungsvorschrift:

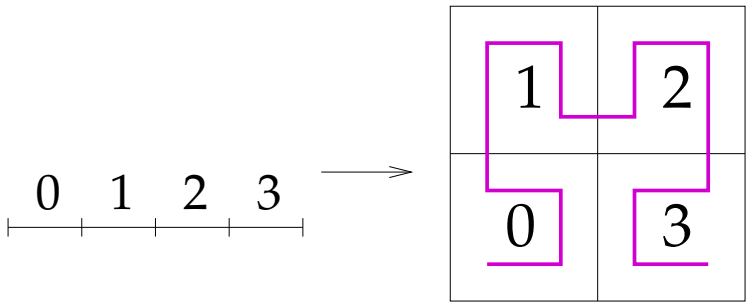


Abbildung 2.3: Intervallschachtelung und Zuordnung der Teilquadrate bei der Konstruktion der Hilbertkurve.

- *Zu jedem* $t \in \mathcal{I} := [0,1]$ *gehört eine Intervallschachtelung*

$$\mathcal{I} \supset [a_1, b_1] \supset \ldots \supset [a_n, b_n] \supset \ldots,$$

  wobei jedes Intervall der Schachtelung durch Viertelung des vorhergehenden entsteht.

- *Jeder solchen Intervallschachtelung ist eindeutig eine 2D-Intervallschachtelung zugeordnet. Die Auswahl der korrekten Teilquadrate geschieht mit Hilfe der zugehörigen Iteration der Hilbertkurve.*

- *Die entstehende 2D-Intervallschachtelung konvergiert eindeutig gegen einen Punkt in* $\mathcal{Q} := [0,1] \times [0,1]$ *– dieser Punkt sei definiert als* $h(t)$.

Das Bild der Abbildung $h : \mathcal{I} \to \mathcal{Q}$ *ist eine raumfüllende Kurve, die* Hilbertkurve.

2.3.4 Beweis: h definiert eine Raumfüllende Kurve

Um nachzuweisen, dass die hier definierte Abbildung h tatsächlich eine Raumfüllende Kurve festlegt, müssen wir beweisen, dass h die folgenden drei Eigenschaften besitzt:

1. h ist eine Abbildung, d.h. jedes $t \in \mathcal{I}$ hat einen *eindeutigen* Funktionswert $h(t)$.

2. $h\colon \mathcal{I} \to \mathcal{Q}$ ist *surjektiv*.

3. h ist *stetig*

Beweis: h definiert eine Abbildung
Die Frage, ob h eine Abbildung definiert, stellt sich als direkte Folge der Tatsache, dass die in der Definition verwendete Intervallschachtelung zum Parameter t nicht eindeutig ist. Sobald nämlich ein Parameter auf einer der Intervallgrenzen liegt, gibt es zwei Möglichkeiten, die Schachtelung fortzusetzen. Wir müssen also zeigen, dass die Definition unabhängig von der Wahl der Intervallschachtelung immer zum gleichen Wert $h(t)$ führt. Der strikte Beweis dazu ist einigermaßen lästig. Wir werden in einem der folgenden Kapitel darauf zurückkommen, wenn wir eine dafür besser geeignete Darstellung der Hilbertkurve besprechen.

Als Rechtfertigung können wir einstweilen den Beweis der Stetigkeit von h anführen. Die beiden Möglichkeiten der Intervallschachtelungen liefern so etwas wie einen linksseitigen und einen rechtsseitigen Grenzwert für $h(t)$ an der Stelle t. Die Stetigkeit von h stellt sicher, dass diese beiden Grenzwerte gleich sind.

Beweis: h ist surjektiv
Für die Surjektivität müssen wir zeigen, dass es für jeden Punkt q des Einheitsquadrats mindestens einen Parameter t gibt, so dass $h(t) = q$. Dies beweisen wir durch Umkehren der Intervallschachtelung:

- Zu jedem Punkt $q \in \mathcal{Q}$ lässt sich eine geeignete 2D-Intervallschachtelung konstruieren, deren 2D-Intervalle gerade die Teilquadrate der rekursiven Konstruktion der Hilbertkurve sind.
- Dieser 2D-Schachtelung ist durch die Konstruktion eindeutig eine Intervallschachtelung im Parameterintervall $\mathcal{I}$ zugeordnet. Aufgrund der Vollständigkeit der reellen Zahlen definiert diese Intervallschachtelung eindeutig einen Parameter t.
- Durch Verwenden dieser beiden Intervallschachtelungen in der Definition der Hilbertkurve erhalten wir leicht, dass $h(t) = q$. Also ist h surjektiv.

An dieser Stelle greifen wir noch einmal das Problem der Eindeutigkeit der Intervallschachtelungen auf. Bei der Definition der Hilbertkurve hätten wir problemlos eine eindeutige Intervallschachtelung verwenden können, und zwar mit Hilfe halboffener Intervalle:

$$\left[a, a+\frac{h}{4}\right), \left[a+\frac{h}{4}, a+\frac{2h}{4}\right), \left[a+\frac{2h}{4}, a+\frac{3h}{4}\right), \left[a+\frac{3h}{4}, a+h\right].$$

Bei einer Intervallschachtelung mit halboffenen Grenzen ist jedoch nicht sichergestellt, dass sie sich gegen einen Grenzwert innerhalb der Intervalle zusammenzieht. Falls sich die Schachtelung nämlich gegen eine offene Intervallgrenze zusammenzieht, dann liegt der gefundene Parameter außerhalb fast aller Intervalle der Schachtelung. Das Verwenden eindeutiger Intervallschachtelung verursacht also mindestens ebenso viele technische Probleme, wie die jetzige Definition.

2.3.5 Stetigkeit der Hilbertkurve

Nachdem wir gezeigt haben, dass unsere Abbildung h das Parameterintervall surjektiv auf das Einheitsquadrat abbildet – also *raumfüllend* ist –, bleibt uns nur noch zu zeigen, dass h auch eine Kurve ist. Dazu müssen wir die *Stetigkeit* von h zeigen. Wir werden im Folgenden sogar eine etwas stärkere Eigenschaft von h zeigen, nämlich dass h *gleichmäßig stetig* ist.

Eine Funktion $f\colon \mathcal{I} \to \mathbb{R}^n$ ist *gleichmäßig stetig* auf einem Intervall $\mathcal{I}$, wenn es zu jedem $\epsilon > 0$ ein $\delta > 0$ gibt, so dass für alle $t_1, t_2 \in \mathcal{I}$ mit $|t_1 - t_2| < \delta$ gilt, dass $\|f(t_1) - f(t_2)\|_2 < \epsilon$.

Wir können dies als „Stetigkeitsspiel“ auffassen: unser Gegner wirft uns ein $\epsilon > 0$ zu. Unsere Aufgabe ist es, ein geeignetes δ zu finden, so dass für zwei beliebige t_1, t_2, die weniger als δ voneinander entfernt sind, die zugehörigen Bilder $f(t_1)$ und $f(t_2)$ einen Abstand von weniger als ϵ haben.

Zum Vergleich: bei der „normalen“ Stetigkeit zeigt man üblicherweise die Stetigkeit in jedem Punkt t_1 des Intervalls $\mathcal{I}$ einzeln. Dabei dürfen wir das δ auch abhängig von t_1 wählen!

Der Beweis der Stetigkeit

Unsere Beweisidee ist, dass wir eine Strategie entwickeln, dieses „Stetigkeitsspiel“ immer zu gewinnen. Dazu versuchen wir für zwei beliebige t_1, t_2 abzuschätzen, wie groß der Abstand $\|h(t_1) - h(t_2)\|_2$ ihrer Bilder maximal werden kann (in Abhängigkeit von $|t_1 - t_2|$). Diese Abschätzung verläuft in vier Schritten:

1. Gegeben seien $t_1, t_2 \in \mathcal{I}$; wir wählen ein $n \in \mathbb{N}$ so, dass $|t_1 - t_2| < 4^{-n}$.

2. Wir betrachten nun die n-te Iteration der Intervallschachtelungen; in dieser haben alle Intervalle die Länge 4^{-n}. Daher überlappt das Intervall $[t_1, t_2]$ maximal zwei Intervalle, die außerdem benachbart sein

müssen (läge ein komplettes Intervall zwischen t_1 und t_2, dann wäre ihr Abstand ja größer als 4^{-n}).

3. Gemäß Konstruktion der Hilbertkurve werden diese beiden Intervalle auf zwei benachbarte Quadrate der Seitenlänge 2^{-n} abgebildet. Entsprechend liegen die beiden Funktionswerte $h(t_1)$ und $h(t_2)$ in einem Rechteck, das durch die beiden benachbarten Quadrate gebildet wird.

4. Dieses Rechteck hat die Seitenlängen 2^{-n} und $2 \cdot 2^{-n}$. Die Länge seiner Diagonale ist demnach $2^{-n} \cdot \sqrt{5}$ (nach dem Satz des Pythagoras). Da beide Funktionswerte $h(t_1)$ und $h(t_2)$ innerhalb des Rechtecks liegen, kann ihr Abstand nicht größer sein als die Diagonale, also gilt: $\|h(t_1) - h(t_2)\|_2 \leq 2^{-n}\sqrt{5}$.

Zu einem gegebenen $\epsilon > 0$ können wir stets ein n finden, so dass $2^{-n}\sqrt{5} < \epsilon$. Mit diesem n finden wir auch ein $\delta := 4^{-n}$, so dass für alle t_1, t_2 mit $|t_1 - t_2| < \delta$ gemäß der obigen Herleitung gilt:

$$\|h(t_1) - h(t_2)\|_2 \leq 2^{-n}\sqrt{5} < \epsilon.$$

Damit ist die Stetigkeit von h gezeigt!

Wir wollen an dieser Stelle festhalten, dass wir nur zwei wesentliche Eigenschaften für den Beweis der Stetigkeit benötigt haben, nämlich:

- dass h zwei benachbarte Intervalle stets auf zwei benachbarte Quadrate abbildet, und

- dass die Intervalle bzw. Quadrate jeweils durch Viertelung gebildet werden. Dies stellt insbesondere den exponentiellen Abfall der Intervall- und Seitenlänge sicher. Außerdem bewirkt es, dass alle Intervalle und Quadrate einer Iteration gleich groß sind.

Wir werden auf diese Beobachtung zurückkommen, wenn wir die Stetigkeit anderer raumfüllender Kurven beweisen müssen.

2.3.6 Die Hilbertkurve nach Moore

Wie in Abbildung 2.4 dargestellt, lassen sich vier geeignet rotierte Hilbertkurven so zusammenfügen, dass wir eine raumfüllende Kurve erhalten,

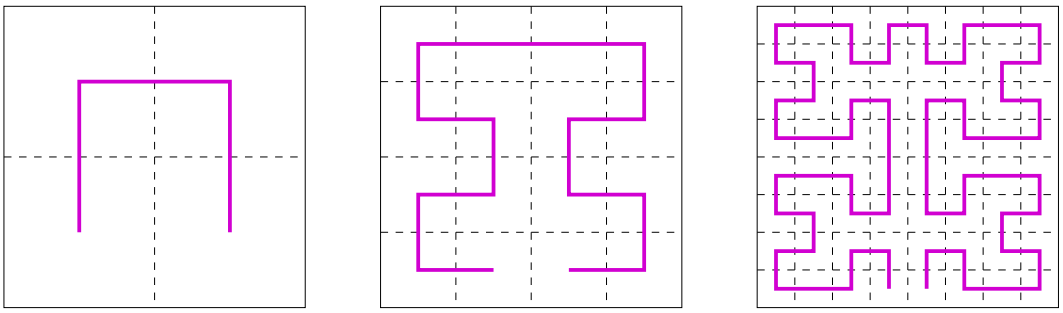


Abbildung 2.4: Die ersten drei Iterationen der Hilbertkurve nach Moore

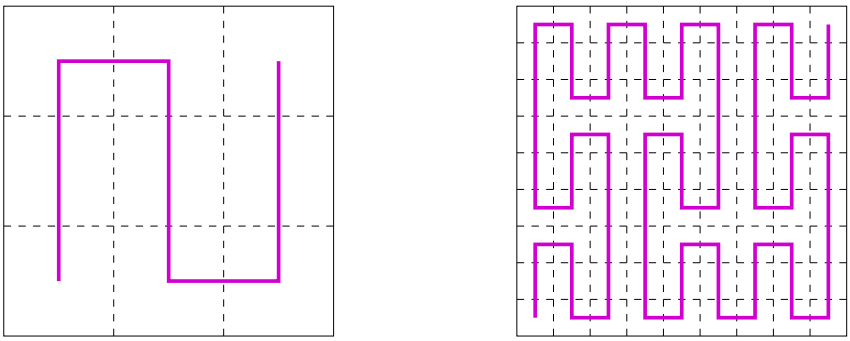


Abbildung 2.5: Die ersten beiden Iterationen der Peanokurve

deren Start- und Endpunkt im Punkt $\left(0, \frac{1}{2}\right)$ zusammenfallen. Die entstehende raumfüllende Kurve wird als *Hilbert-Moore-Kurve* bezeichnet.

Bei der Konstruktion der Iterationen der Hilbert-Moore-Kurve müssen wir lediglich im ersten Schritt eine geänderte Orientierung bei der Transformation der Teilkurven beachten.

2.4 Konstruktion der Peanokurve

Nach beinahe dem gleichem Muster wie bei der Hilbertkurve können wir auch die geschichtlich erste raumfüllende Kurve konstruieren – die *Peanokurve*. Wir entwickeln eine rekursive Konstruktion von Iterationen der Peanokurve und definieren die Peanokurve als Grenzwert dieses Prozesses.

Die rekursive Konstruktion erfolgt für die Peanokurve wie folgt:

1. Statt in vier Teilquadrate teilen wir das quadratisches Grundgebiet bei der Peanokurve in *neun* Teilquadrate auf – dabei reduziert sich die Seitenlänge der Teilquadrate jeweils auf ein Drittel der ursprünglichen.

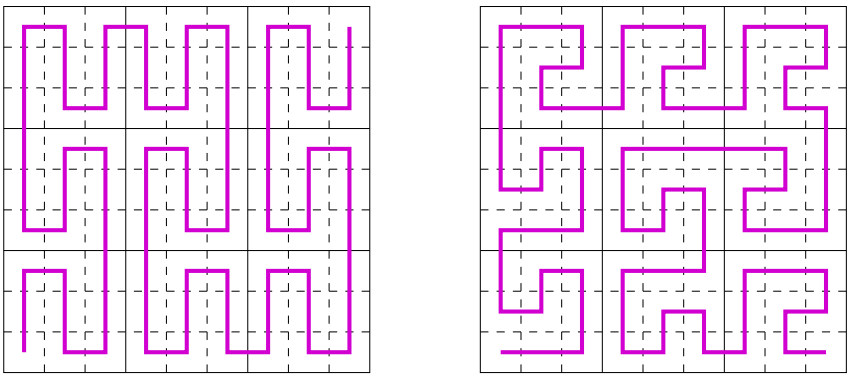
2. Entsprechend entwickeln wir für das Parameterintervall Intervallschachtelungen, die auf einer fortgesetzten Neunteilung der Intervalle beruhen. Jedes Teilintervall soll durch die Peanoabbildung auf ein bestimmtes Teilquadrat abgebildet werden.

3. Jedes Teilquadrat enthält also eine geeignet transformierte Peanokurve. Die gesamte Kurve entsteht wieder durch Zusammensetzen dieser Teilkurven. Gemäß Abbildung 2.5 setzen wir die Teilkurven so zusammen, dass sich die für die Peanokurve typische Schlängelung ergibt (hier eine senkrechte Schlängelung).

4. Die notwendigen Transformation können wir ebenfalls aus Abbildung 2.5 entnehmen. Wir sehen, dass wir im Gegensatz zur Hilbertkurve keine Drehungen benötigen – wir benötigen lediglich Spiegelungen an der vertikalen bzw. horizontalen Achse. Die Spiegelung an der horizontalen Achse benötigen wir insbesondere für die korrekte Laufrichtung der Kurve: in den mittleren drei Teilquadraten muss die Kurve statt von unten nach oben jeweils von oben nach unten laufen.

Die Richtung der „Schlängelung“ kann ebenso vertikal (wie in unserer Definition) oder horizontal erfolgen – dann ergibt sich zum Beispiel eine um 90 Grad gedrehte Peanokurve. Die Schlängelung kann jedoch auch abwechselnd oder gemischt in waagrechter und senkrechter Richtung erfolgen. Dadurch lassen sich viele verschiedene Peanokurven erzeugen – es gibt in der Tat *272 verschiedene* Varianten der Peanokurve. Zusätzlich gibt es noch die sogenannten *Peano-Meander-Kurven* – deren erste Iteration ist in Abbildung 2.6b dargestellt.

Dagegen gibt es von der zweidimensionalen Hilbertkurve tatsächlich nur zwei Varianten, die sich nicht durch Spiegelung oder Rotation ineinander überführen lassen – nämlich die „normale“ Hilbertkurve und die Hilbertkurve nach Moore.



(a) Peanokurve (b) Peano-Meander-Kurve

Abbildung 2.6: Zwei raumfüllende Kurven des Peanotyps

Beweis von Stetigkeit und Surjektivität
Um nachzuweisen, dass durch obige Konstruktion eine raumfüllende Kurve definiert wird, müssen wir wieder insbesondere die Stetigkeit und die Surjektivität der entsprechenden Abbildung nachweisen. Die entsprechenden Beweise für die Hilbertkurve können dazu beinahe wörtlich übernommen werden:

- die *Surjektivität* ergibt sich aus den synchron laufenden Intervall- und Teilquadratschachtelungen: zu jedem Punkt des Einheitsquadrats existiert eine geeignete Schachtelung von Teilquadraten. Die zugehörigen Intervalle liefern eine Intervallschachtelung gegen das Urbild des Punktes, das damit insbesondere für jeden Punkt existiert.

- beim Beweis der Stetigkeit der Hilbertkurve hatten wir bereits festgehalten, dass wir nur zwei wesentliche Eigenschaften zum Beweis benötigt hatten:

  1. dass zwei benachbarte Intervalle stets auf zwei benachbarte Quadrate abbildet werden, und
  2. dass die Intervalle bzw. Quadrate durch rekursives Unterteilen in gleich große Teilintervalle bzw. -quadrate gebildet werden.

  Beide Bedingungen sind für die Peanokonstruktion gegeben, so dass sich der Beweis der Stetigkeit ebenfalls einfach übertragen lässt.

2.5 Raumfüllende Kurven – Benötigte Algorithmen

Unser primäres Ziel ist die Verwendung raumfüllender Kurven zur Linearisierung von Daten. Da wir die raumfüllenden Kurven nun vorgestellt haben und sozusagen nachgewiesen haben, dass es sie tatsächlich gibt, können wir uns für den nächsten Schritt eine Art Pflichtenheft erstellen. Welche Operationen werden wir für das Arbeiten mit raumfüllenden Kurven benötigen?

Die ersten beiden Operationen dienen dem lesenden und schreibenden Zugriff auf die Daten. Wir müssen einem Datum seinen Index zuordnen können, und umgekehrt:

Berechnung des Funktionswerts:
Zu einem gegebenen Index $t \in \mathcal{I}$ ist der zugehörige Punkt $h(t)$ auf der raumfüllenden Kurve gesucht. Das heißt, wir suchen zu einem Index das zugehörige Datum.

Berechnung des Index eines Punktes:
Gegeben ist hier ein Punkt $p \in \mathcal{Q}$; gesucht ist ein Parameter t, so dass $h(t) = p$. Wesentliches Problem ist hier, dass die Umkehrung der Abbildung h nicht eindeutig ist – die Abbildung h ist nicht bijektiv! Daher werden wir hier eine technisch eindeutig gemachte Umkehrabbildung $\bar{h}^{-1}$ verwenden müssen.

Als Beispiel einer häufig benötigten Operation auf den linearisierten Daten werden wir uns zunächst die folgende vornehmen:

Traversierung der durch h indizierten Objekte:
Eine gegebene Menge von Punkten $p_i \in \mathcal{Q}$ – im Allgemeinen sind das „Positionen“ von Objekten im mehrdimensionalen Raum – ist so zu durchlaufen, dass

$$\bar{h}^{-1}(p_{i_0}) < \bar{h}^{-1}(p_{i_1}) < \dots$$

Einfacher gesagt wollen wir die Daten in der Reihenfolge ihrer Indizes durchlaufen.

Eine ganz ähnlich Aufgabe ist übrigens das *traveling salesman* Problem, bei dem die Positionen so abzulaufen sind, dass der Laufweg am kürzesten

ist (raumfüllende Kurven liefern in der Tat eine gute heuristische Lösung für dieses Problem). Eine übliche Anwendung der Traversierung wäre etwa, die Farbe jedes einzelnen Pixels eines Bildes auf bestimmte Art zu ändern.

Sowohl zur Berechnung von Funktionswerten bzw. Indizes, als auch für die Traversierung benötigen wir vor allem bessere Möglichkeiten zur mathematischen Beschreibung der raumfüllenden Kurven. Die bisher verwendete Definition über die Intervallschachtelung ist dafür zu unhandlich. Zudem ist die Konstruktion der Kurven durch die Iterationen und approximierenden Polygone bei weitem nicht exakt genug beschrieben, als dass wir sie als Programm einem Computer überlassen könnten. Wir werden in den beiden folgenden Kapiteln zwei Möglichkeiten zur Beschreibung raumfüllender Kurven kennen lernen.

3 Beschreibung Raumfüllender Kurven durch Grammatiken

3.1 Darstellung der Hilbertkurve durch eine Grammatik

Die Konstruktion der Iterationen der Hilbertkurve geschah dadurch, dass wir beim Aufteilen in die vier Teilquadrate jedes Teilquadrat wieder mit der vorherigen Iteration der Hilbertkurve gefüllt haben – eine geeignete Rotation bzw. Spiegelung der Kurve vorausgesetzt.

Wir werden nun untersuchen, welche Art rotierter oder gespiegelter Kurven überhaupt vorkommen. Dazu markieren wir die in den Iterationen auftretenden *Grundmotive*. Ein Grundmotiv ist dabei jeweils ein Teilstück, das entweder der 0-ten Iteration (also *„hoch-rechts-runter“*) entspricht, oder aus Rotation bzw. Spiegelung aus diesem hervorgeht. Wir erhalten das in Abbildung 3.1 gezeigte Schema.

Wir sehen, dass sich die Iterationen aus lediglich vier Grundmotiven zusammensetzen lassen – in Abbildung 3.1 gekennzeichnet durch die Buchstaben H, A, B und C. Die zu diesen Buchstaben gehörigen Grundmotive

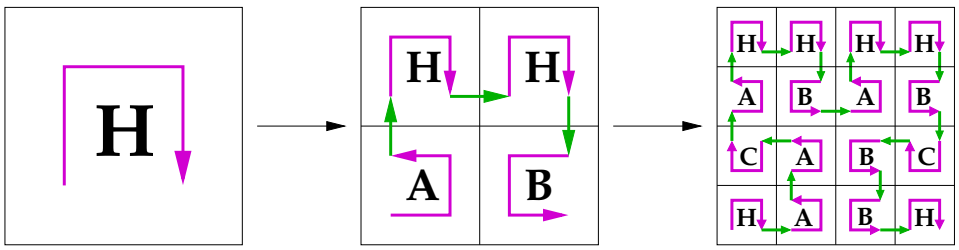


Abbildung 3.1: Identifikation der Grundmotive in den ersten drei Iterationen der Hilbertkurve

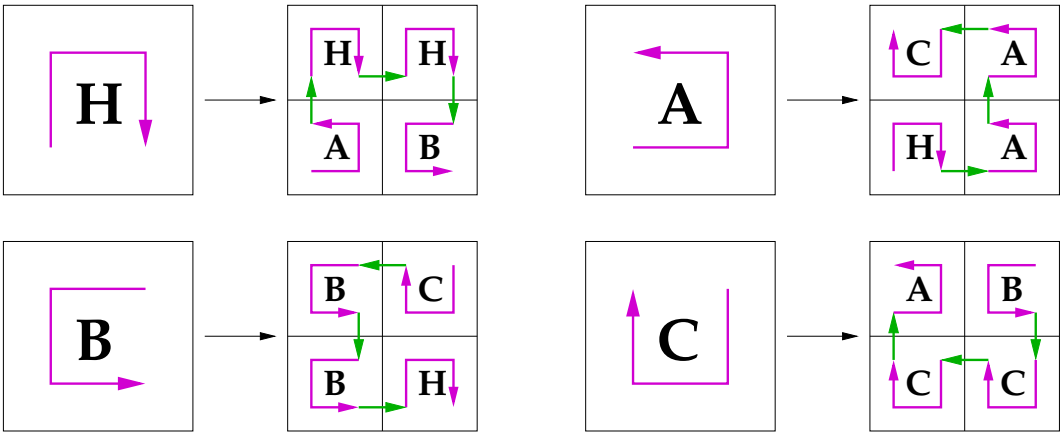


Abbildung 3.2: Ersetzungsschema der Grundmotive bei der Hilbertkurve

werden in der nächsten Iteration jeweils durch eine erste Iteration der Hilbertkurve ersetzt. Dies geschieht nach einem festen Ersetzungsschema, das in Abbildung 3.2 dargestellt ist.

Diesem Ersetzungsschema können wir auch entnehmen, dass tatsächlich keine neuen Grundmotive hinzukommen. Das Schema ist abgeschlossen. Wir werden diese sukzessive Erstellung von Iterationen im Folgenden durch eine Art *Grammatik* beschreiben.

Eine Grammatik für die Iterationen der Hilbertkurve

Unsere Grammatik sei gegeben durch

1. eine Menge von **Nicht-Terminal-Zeichen**: $\{H, A, B, C\}$. Die Nicht-Terminal-Zeichen repräsentieren die vier verschiedenen Grundmotive. Das Grundmotiv H ist als **Startsymbol** ausgezeichnet.

2. eine Menge von **Terminal-Zeichen**: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. Die Terminal-Zeichen beschreiben jeweils die Übergänge zwischen den Grundmotiven. In Abbildung 3.2 sind sie durch die grünen Pfeile repräsentiert.

3. eine Menge von **Produktionen**, d.h. von Ersetzungsregeln:

$$\begin{aligned} H &\longleftarrow A \uparrow H \rightarrow H \downarrow B \\ A &\longleftarrow H \rightarrow A \uparrow A \leftarrow C \\ B &\longleftarrow C \leftarrow B \downarrow B \rightarrow H \\ C &\longleftarrow B \downarrow C \leftarrow C \uparrow A \end{aligned}$$

   Die Ersetzungsregeln legen fest, welches Grundmotiv jeweils durch welche Abfolge von Grundmotiven und Übergängen ersetzt werden muss, um die nächste Iteration zu erhalten.

4. eine **Ersetzungsregel**: in einem Wort dürfen stets nur **alle Nicht-Terminal-Zeichen gleichzeitig** ersetzt werden. Damit wird in jeder Verfeinerungsstufe die gleiche Verfeinerungstiefe in jedem Teilquadrat erzwungen.

Ohne die zusätzliche Ersetzungsregel entspräche unsere Grammatik einer ganz gewöhlichen kontextfreien Grammatik, wie sie in der Informatik oft verwendet wird. Mit der Ersetzungsregel jedoch ist die Grammatik den sogenannten **L-Systemen** verwandt.

Satzformen der Grammatik
Welche Wörter (in diesem Zusammenhang sprechen wir besser von *Satzformen*) werden nun von unserer Grammatik erzeugt? Eine Antwort darauf erhalten wir, indem wir die Ersetzungsregel (wiederholt) auf das Startsymbol anwenden:

$$\begin{aligned} H &\longleftarrow A \uparrow H \rightarrow H \downarrow B \\ &\longleftarrow H \rightarrow A \uparrow A \leftarrow C \uparrow A \uparrow H \rightarrow H \downarrow B \rightarrow A \uparrow H \rightarrow H \downarrow B \downarrow C \leftarrow B \downarrow B \rightarrow H \end{aligned}$$

Wir stellen fest, dass die Pfeile in einfacher Weise die Iterationen der Hilbertkurve in einer Art „Schildkröten-Grafik“ *(Turtle-Grafik)* beschreiben. Unsere Schildkröte befolgt jedoch nicht relative Bewegungsanweisungen – biege nach rechts/links ab, etc. –, wie in der bekannten Version der Turtle-Grafik. Stattdessen hat sie eine Art eingebauten Kompass und kann auf eine „Landkarte“ (nämlich unser Einheitsquadrat, das die Hilbert-Iterationen ausfüllen sollen) schrittweise nach oben, unten, rechts oder links marschieren.

Abgeschlossenheit der Grammatik
Die Grammatik für die Hilbertkurve liefert uns als Nebenresultat noch die Antwort auf eine Frage, die wir bisher noch gar nicht gestellt haben – aber hätten stellen (und beantworten) sollen. Diese Frage ist, ob das System der Transformationen und des Zusammensetzens der Teiliterationen zu den Iterationen der Hilbertkurve überhaupt so beschaffen ist, dass wir es unendlich fortsetzen können.

Die entwickelte Grammatik beantwortet uns diese Frage. Wir erkennen erstens, dass tatsächlich nur vier Grundmuster in den Iterationen auftreten können – nämlich die vier, die durch die Nichtterminale H, A, B und C repräsentiert werden. Zweitens erkennen wir an den Terminalzeichen, dass auch die direkte Verbindung der Teiliterationen gewährleistet ist.

Nicht zuletzt liefert die Grammatik auch endlich eine mathematisch exakte Beschreibung der Abfolge der notwendigen Transformationen – unsere bisherige Beschreibung über das Zeichnen der ersten paar Iterationen konnte nur für die ersten Schritte befriedigend sein.

3.2 Ein Algorithmus zur Traversierung zweidimensionaler Daten

Die Iterationen der Hilbertkurve liefern eine Vorschrift, in welcher Reihenfolge die Teilquadrate zur Konstruktion der Hilbertkurve abgelaufen werden müssen. Für zweidimensionale Daten, die einer Art Rasterung entstammen, können wir die Iterationen direkt als Traversierung übernehmen. Solche Daten sind zum Beispiel:

- Bilddaten, wenn wir davon ausgehen, dass ein Bild aus rechteckigen Pixeln zusammengesetzt ist;
- Matrizen in der linearen Algebra;
- generell alle Daten, die in einem zweidimensionalen Array abgespeichert werden.

Der Algorithmus zur Traversierung muss nun lediglich die Satzformen der Grammatik sukzessive entwickeln und die durch die Pfeile vorgegebenen Bewegungen in der korrekten Reihenfolge durchführen. Da der Ab-

leitungsbaum der Satzformen gerade einem Aufrufbaum rekursiver Unterprogramme entspricht, werden wir die Nicht-Terminalzeichen H, A, B und C mit entsprechenden rekursiven Prozeduren bzw. Methoden identifizieren. Die Terminalzeichen $\uparrow$, $\downarrow$, $\leftarrow$ und $\rightarrow$ ersetzen wir durch die Aufrufe von Prozeduren, die in der jeweiligen Datenstruktur einem Schritt nach oben, unten, links oder rechts entsprechen.

Das Nicht-Terminalzeichen H und die zugehörige Produktion $H \longleftarrow A \uparrow H \rightarrow H \downarrow B$ übersetzen wir demnach in eine Prozedur `H`, wie sie in Algorithmus 1 formuliert ist.

Algorithmus 1 : Traversierung entlang einer Hilbertkurve

```
proc H(tiefe) begin
    if tiefe > 0 then
        A(tiefe-1); up();
        H(tiefe-1); right();
        H(tiefe-1); down();
        B(tiefe-1);
    end
end proc
```

Die Prozeduren `A()`, `B()` und `C()` ergeben sich ganz analog aus den anderen Nicht-Terminalzeichen und Produktionen. Die Prozeduren `up()`, `down()`, etc. implementieren die elementaren Bewegungen auf der zu traversierenden Datenstruktur.

3.3 Grammatiken für Peanokurven

Wie für die Hilbertkurve lässt sich auch für die Peanokurve (vgl. Abbildung 3.3) eine Grammatik-Darstellung herleiten.

Der Bezug zwischen Grammatik und Konstruktionsprinzip ist in Abbildung 3.4 verdeutlicht. Man beachte vor allem, dass sich die Bausteine P und R, ebenso wie Q und S, nur in der Durchlaufrichtung unterscheiden.

Die Grammatik bilden wir analog zu der der Hilbertkurve:

- Als Menge der Nicht-Terminalzeichen definieren wir $\{P, Q, R, S\}$, wobei die Symbole wieder die in Abbildung 3.4 aufgeführten Grund-

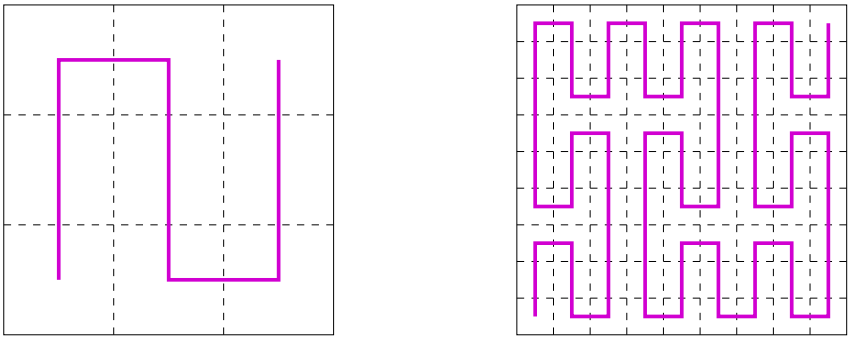


Abbildung 3.3: Die ersten beiden Iterationen der Peanokurve

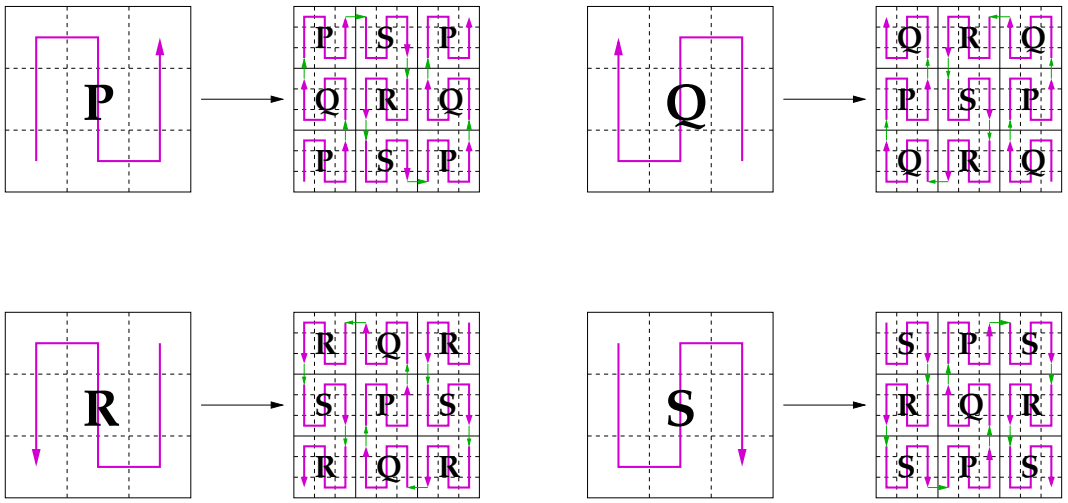


Abbildung 3.4: Bezug zwischen Grammatik und Konstruktion bei der Peanokurve

muster repräsentieren. Startsymbol der Grammatik ist das Zeichen P.

- Die Menge der Terminalzeichen ist gegenüber der Hilbertkurve unverändert: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$.
- Abbildung 3.4 entnehmen wir dann folgende Produktionen:

$$\begin{aligned} P &\longleftarrow P \uparrow Q \uparrow P \rightarrow S \downarrow R \downarrow S \rightarrow P \uparrow Q \uparrow P \\ Q &\longleftarrow Q \uparrow P \uparrow Q \leftarrow R \downarrow S \downarrow R \leftarrow Q \uparrow P \uparrow Q \\ R &\longleftarrow R \downarrow S \downarrow R \leftarrow Q \uparrow P \uparrow Q \leftarrow R \downarrow S \downarrow R \\ S &\longleftarrow S \downarrow R \downarrow S \rightarrow P \uparrow Q \uparrow P \rightarrow S \downarrow R \downarrow S \end{aligned}$$

Als Ersetzungsregel legen wir wieder fest, dass in jeder Satzform stets nur alle Nicht-Terminalzeichen *gleichzeitig* ersetzt werden dürfen.

Aus der Grammatik können wir wieder einen Algorithmus für die Traversierung eines Datensatzes gewinnen. Die dem Symbol P zugeordnete Prozedur `P()` ist in Algorithmus 2 beschrieben.

Algorithmus 2 : Traversierung entlang einer Peanokurve

```
proc P(tiefe) begin
    if tiefe > 0 then
        P(tiefe-1); up();
        Q(tiefe-1); up();
        P(tiefe-1); right();
        S(tiefe-1); down();
        R(tiefe-1); down();
        S(tiefe-1); right();
        P(tiefe-1); up();
        Q(tiefe-1); up();
        P(tiefe-1);
    end
end proc
```

4 Arithmetisierung Raumfüllender Kurven

Bis jetzt haben wir lediglich die endlichen Iterationen der Hilbertkurve und der Peanokurven untersucht. Diese können uns zwar eine ungefähre Vorstellung der „unendlichen“ Kurven vermitteln, jedoch haben wir bis jetzt für noch keinen einzigen Parameter den zugehörigen Punkt der Kurve berechnet. Die Darstellung der Kurven durch die Grammatiken hat dazu auch nicht das geeignete Werkzeug geliefert, da die Iterationen immer als ganzes erzeugt wurden.

In diesem Kapitel werden wir daher eine Systematik entwickeln, die es erlaubt die Hilbertkurve, wie auch die Peanokurven und alle verwandten Kurven arithmetisch zu beschreiben.

4.1 Arithmetisierung der Hilbertabbildung

Die wesentlichen Konstruktionsideen für die Definition der Hilbertkurve waren:

1. Das Approximieren eines gegebenen Parameters durch eine Intervallschachtelung des Parameter-Intervalls $\mathcal{I}$, wobei die Intervalle jeweils einer Viertelung des vorhergehenden Intervalls entstammen.

2. Das rekursive Unterteilen des quadratischen Zielgebiets in Teilquadrate, die jeweils durch eine skalierte, transponierte und ggf. gespiegelte/rotierte Hilbertkurve gefüllt werden.

Die *Arithmetisierung* der Hilbertabbildung setzt diese beiden Ideen in eine formelle, mathematische Beschreibung um.

Quarternärdarstellung des Parameters
Die erforderliche Intervallschachtelung können wir geeignet durch die Darstellung des Parameters als Quarternärzahl wiedergeben. Sehen wir uns

dazu die typischen Intervallgrenzen im Quarternärsystem an:

$$\left[0, \frac{1}{4}\right] = [0_4.0, 0_4.1]\,, \qquad \left[\frac{1}{4}, \frac{2}{4}\right] = [0_4.1, 0_4.2]\,,$$
$$\left[\frac{2}{4}, \frac{3}{4}\right] = [0_4.2, 0_4.3]\,, \qquad \left[\frac{3}{4}, 1\right] = [0_4.3, 1_4.0]\,.$$

Die Koeffizienten der Quarternärdarstellung geben gerade die Nummer des zu wählenden Viertelintervalls an. Zum Beispiel erhalten wir für den Parameter $t = \frac{2}{5}$ die Intervallschachtelung

$$[0, 1]\,, [0_4.1, 0_4.2]\,, [0_4.12, 0_4.13]\,, [0_4.121, 0_4.122]\,, \ldots,$$

entsprechend der Quarternärdarstellung $\frac{2}{5} = 0_4.121212\ldots$

Abbildung der rekursiven Teilquadrate auf das Einheitsquadrat

Die zweite wesentliche Idee zur Konstruktion der Hilbertkurve ist das rekursive Zurückführen des Problems auf die vier Teilquadrate. Unserer Vorstellung nach befindet sich in jedem der vier Teilquadrate eine entsprechend skalierte und transformierte Hilbertkurve.

Unserem rekursiven Ansatz folgend, nehmen wir an, wir hätten für die Teilhilbertkurve den Kurvenpunkt bereits berechnet. Dieser ist natürlich bezogen auf das Einheitsquadrat $[0, 1]^2$ berechnet. Wir müssen daher die relative Lage des Punktes im entsprechenden Teilquadrat ermitteln.

Dazu müssen wir für jedes Teilquadrat einen Transformationsoperator angeben, der das Einheitsquadrat in das jeweilige Teilquadrat abbildet. Welcher Operator angewendet werden muss, verrät uns die erste Quarternärstelle. Sie legt das Intervall der Intervallschachtelung und damit auch das Teilquadrat fest. Wir bezeichnen die Operatoren gemäß Abbildung 4.1 als H_0, H_1, H_2 und H_3.

$$H_0 := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} \frac{1}{2}y \\ \frac{1}{2}x \end{pmatrix} \qquad H_1 := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} \frac{1}{2}x \\ \frac{1}{2}y + \frac{1}{2} \end{pmatrix}$$

$$H_2 := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} \frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \end{pmatrix} \qquad H_3 := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} -\frac{1}{2}y + 1 \\ -\frac{1}{2}x + \frac{1}{2} \end{pmatrix}$$

Alle Operatoren skalieren zunächst den übergebenen Vektor mit dem Faktor $\frac{1}{2}$, da die Teilquadrate jeweils halbe Seitenlänge haben. Zudem wer-

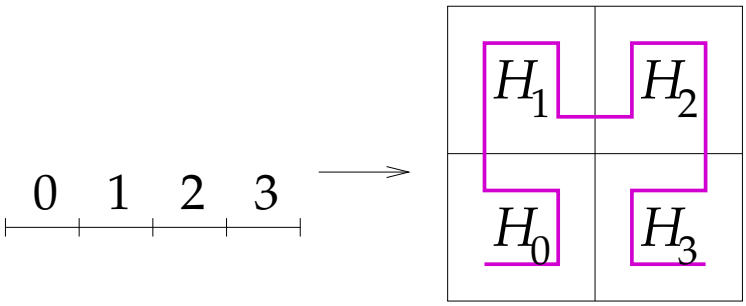


Abbildung 4.1: Intervallschachtelung und Zuordnung der Transformationsoperatoren $H_0, \ldots, H_3$ (vgl. Abbildung 2.3 auf Seite 16).

den von den einzelnen Operatoren je nach Situation Rotationen bzw. Spiegelungen sowie eine Translation durchgeführt:

- der Operator H_0 spiegelt sein Argument an der Hauptdiagonalen (vertauscht x- und y-Komponente). Dies entspricht gerade der gewünschten Vierteldrehung im Uhrzeigersinn inklusive Umkehrung der Durchlaufrichtung.

- die Operatoren H_1 und H_2 behalten die Orientierung der Kurve bei, daher ist keine Rotation oder Spiegelung nötig. Jedoch wird die Kurve entsprechend skaliert in je eines der „oberen“ Teilquadrate verschoben. H_1 verschiebt die Kurven um $\frac{1}{2}$ in y-Richtung, also ins linke, obere Teilquadrat. H_2 verschiebt die Kurven entsprechend diagonal ins rechte obere Teilquadrat.

- der Operator H_3 benötigt zusätzlich zur Vertauschung von x- und y-Komponente (siehe Operator H_0) auch noch je eine Spiegelung sowohl in x- als auch in y-Richtung (daher jeweils Skalierung mit $-\frac{1}{2}$). Die Spiegelung an der vertikalen Achse ist direkt ersichtlich. Die Notwendigkeit, zusätzlich an der horizontalen Achse zu spiegeln, erkennt man erst, wenn man die Laufrichtung der Kurve beachtet: sie startet oben rechts im Teilquadrat und endet in der rechten unteren Ecke. Die Lage des Ursprungs der Teilkurve rechts oben im Teilquadrat erfordert außerdem die Verschiebung des Kurvenstarts (gleichbedeutend mit der der gesamten Kurve) um den Vektor $\begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}$.

In Matrix-Schreibweise lauten die Operatoren $H_0, \ldots, H_3$ wie folgt:

$$H_0 := \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad H_1 := \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{2} \end{pmatrix}$$

$$H_2 := \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \quad H_3 := \begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}$$

4.2 Berechnung der Funktionswerte von h

Die Berechnung eines Funktionswerts $h(t)$ läuft dann in drei Schritten ab:

1. Berechne die Quarternärdarstellung des Parameters: $t = 0_4.q_1q_2q_3q_4\ldots$ Damit ist klar, dass t im q_1-ten Intervall und $h(t)$ im q_1-ten Teilquadrat liegt.

2. Im q_1-ten Teilquadrat entspricht der Parameter t dem lokalen Parameter $\tilde{t} = 0_4.q_2q_3q_4\ldots$ – in der Quarternärdarstellung ist lediglich die erste Stelle wegzulassen. Berechne das Bild von $\tilde{t}$ im q_1-ten Teilquadrat relativ zu diesem Teilquadrat, also gerade $h(\tilde{t})$.

3. Rechne die lokalen Koordinaten von $h(\tilde{t})$ im Teilquadrat auf die Koordinaten im Ausgangsquadrat um. Dies geschieht mit Hilfe der Transformation H_{q_1}.

Die wesentliche Idee zur Arithmetisierung der Hilbertkurve lässt sich somit in folgender Rekursionsgleichung zusammenfassen:

$$h(0_4.q_1q_2q_3q_4\ldots) = H_{q_1} \circ h(0_4.q_2q_3q_4\ldots). \tag{4.1}$$

Durch sukzessives Anwenden der Rekursionsgleichung erhalten wir

$$\begin{aligned} h(0_4.q_1q_2q_3q_4\ldots) &= H_{q_1} \circ h(0_4.q_2q_3q_4\ldots) \\ &= H_{q_1} \circ H_{q_2} \circ h(0_4.q_3q_4\ldots) \\ &= \ldots \end{aligned}$$

Wie wird nun diese Rekursion abgebrochen? Dazu betrachten wir den Fall einer endlichen Quarternärdarstellung. In einer endlichen Quarternärzahl

folgen ab einer Stelle q_n nur noch 0-Stellen, so dass gilt:

$$\begin{aligned} h(0_4.q_1q_2\dots q_n) &= h(0_4.q_1q_2\dots q_n000\dots) \\ &= H_{q_1} \circ H_{q_2} \circ \dots \circ H_{q_n} \circ \underbrace{h(0_4.000\dots)}_{=h(0)}. \end{aligned}$$

Aus der Konstruktion der Hilbertkurve wissen wir bereits, dass $h(0) = (0,0)$. Aber ist dies auch mit der skizzierten Arithmetisierung konsistent? Für $t = 0 = 0_4.000\dots$ lautet die Rekurisionsgleichung (4.1)

$$\begin{aligned} h(0_4.0000\dots) &= H_0 \circ h(0_4.000\dots) \\ \Leftrightarrow h(0) &= H_0 \circ h(0). \end{aligned}$$

$h(0)$ muss also Fixpunkt von H_0 sein. $h(0) = (0,0)$ ist aber gerade der einzige Fixpunkt von H_0.

Wenn also t ein endlicher Quarternärbruch ist, also $t = 0_4.q_1q_2q_3\dots q_n$, dann lässt sich $h(t)$ darstellen als

$$h(0_4.q_1q_2q_3\dots q_n) = H_{q_1} \circ H_{q_2} \circ H_{q_3} \circ \dots \circ H_{q_n} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{4.2}$$

Beispiel: Berechnung von $h\left(\frac{1}{4}\right)$ und $h\left(\frac{1}{8}\right)$

Zur Berechnung der Hilbertabbildung für die Parameter $t = \frac{1}{4}$ und $t = \frac{1}{8}$ müssen wir zunächst deren Darstellung im Quarternärsystem ermitteln. Wir erhalten:

$$\frac{1}{4} = 0_4.1 \qquad \text{und} \qquad \frac{1}{8} = 0_4.02$$

Damit erhalten wir für $h\left(\frac{1}{4}\right)$:

$$h\left(\frac{1}{4}\right) = H_1 \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{2} \end{pmatrix}$$

Dies entspricht auch gut unserer Vorstellung: nach Ablaufen von einem Viertel des Parameterintervalls sollte die Hilbertkurve auch ein Viertel des Einheitsquadrats gefüllt haben. Der Punkt $h\left(\frac{1}{4}\right)$ sollte also gerade der Übergangspunkt zwischen dem ersten und zweiten Teilquadrat sein. Dieser liegt konstruktionsgemäß in der linken oberen Ecke des ersten Teilquadrats, also auf $\left(0, \frac{1}{2}\right)$.

Analog berechnen wir den Abbildungswert für den Parameter $t = h\left(\frac{1}{8}\right)$:

$$\begin{aligned} h\left(\frac{1}{8}\right) &= H_0 \circ H_2 \begin{pmatrix} 0 \\ 0 \end{pmatrix} = H_0 \left(\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \right) \\ &= H_0 \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \end{aligned}$$

Wieder stimmt dies mit unserer Vorstellung überein: nach einem Achtel des Intervalls sollte die Hälfte des ersten Teilquadrats gefüllt sein. Der Punkt befindet sich in der Mitte des ersten Teilquadrats – dort wo die Hilbertkurve vom zweiten (Unter-)Teilquadrat ins dritte wechselt.

Algorithmus zur Berechnung der Hilbertkurve
Ein Algorithmus zur Berechnung der Funktionswerte von h muss demnach zwei Teilaufgaben lösen:

1. die Berechnung der Quarternärstellen:
   dazu können wir gemäß der Gleichung
   $$4 \cdot 0_4.q_1q_2q_3q_4\ldots = (q_1.q_2q_3q_4\ldots)_4$$
   den Parameter jeweils mit 4 multiplizieren und den Vorkommaanteil abschneiden.

2. die Anwendung der entsprechenden Operatoren H_q in der richtigen Reihenfolge – dazu können wir direkt die Rekursionsgleichung (4.1) in einen rekursiven Algorithmus übernehmen.

Als Abbruchkriterium für die Rekursion können wir zum Beispiel eine feste Rekursionstiefe vorgeben – das entspricht gerade einer festen Zahl von berücksichtigten Quarternärstellen. Das Ergebnis ist die Funktion `hilbert`.

Ebenso könnten wir auch eine Genauigkeitsschranke ϵ als Abbruchkriterium implementieren. Dazu interpretieren wir ϵ als Seitenlänge eines Teilquadrats der gebildeten 2D-Schachtelung, in dem der gesuchte Punkt liegt. Dann ist gerade $\epsilon = 2^{-tiefe}$ und wir erhalten die Funktion `hilbertEps`.

4.3 Zur Eindeutigkeit der Hilbertfunktion

In diesem Abschnitt kommen wir auf eine Frage zurück, die wir schon bei der Definition der Hilbertabbildung gestellt haben:

Funktion `hilbert`

```
func hilbert(t, tiefe) begin
    if tiefe = 0 then
        return (0,0)
    else
        // nächste Quarternärstelle bestimmen
        q := floor(4*t);
        r := 4*t - q;
        (x,y) := hilbert(r,tiefe-1);
        switch q do
            case 0: return ( y/2, x/2) ;
            case 1: return ( x/2, y/2 + 0.5) ;
            case 2: return ( x/2+0.5, y/2+0.5) ;
            case 3: return ( 1.0-y/2, 0.5-x/2) ;
        end
    end
end func
```

Funktion `hilbertEps`

```
func hilbertEps(t, eps) begin
    if eps > 1 then
        return (0,0)
    else
        // nächste Quarternärstelle bestimmen
        q := floor(4*t);
        r := 4*t - q;
        (x,y) := hilbertEps(r,2*eps);
        switch q do
            case 0: return ( y/2, x/2) ;
            case 1: return ( x/2, y/2 + 0.5) ;
            case 2: return ( x/2+0.5, y/2+0.5) ;
            case 3: return ( 1.0-y/2, 0.5-x/2) ;
        end
    end
end func
```

Sind die Funktionswerte von h unabhängig von der Wahl der Quarternärentwicklung im Sinne von

$$h(0_4.q_1 \dots q_n) = h(0_4.q_1 \dots q_{n-1}(q_n - 1)333\dots), \qquad q_n \neq 0$$

Den Fall $q_n = 0$ dürfen wir hier ausschließen, da dann $0_4.q_1 \dots q_n = 0_4.q_1 \dots q_{n-1}$ und unser Problem schlicht eine Stelle nach vorne gezogen wird.

Die beiden unterschiedlichen Quarternärentwicklungen entsprechen den beiden unterschiedlichen Intervallschachtelungen, die wir in Abschnitt 2.3.4 angesprochen haben. Mit der Arithmetisierung haben wir jedoch diesmal ein mathematisches Mittel, die Eindeutigkeit zu zeigen und nicht nur durch die Stetigkeit zu rechtfertigen.

Der Beweis der Unabhängigkeit läuft in zwei Schritten:

1. Berechne den Grenzwert

$$\lim_{n\to\infty} H_3^n \qquad \text{bzw.} \qquad \lim_{n\to\infty} H_3^n \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

2. Zeige für $q_n = 1, 2, 3$, dass

$$H_{q_n} \circ \begin{pmatrix} 0 \\ 0 \end{pmatrix} = H_{q_n - 1} \circ \lim_{n\to\infty} H_3^n \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Berechnung von h für unendliche Quarternärbrüche

Die Berechnung von $\lim_{n\to\infty} H_3^n$ benutzen wir als Beispiel für die Berechnung von h für unendliche Quarternärbrüche im Allgemeinen. Für unseren Beweis genügt es, wenn wir den Wert

$$h(0_4.3333\dots) = H_3 \circ H_3 \circ \dots \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \lim_{n\to\infty} H_3^n \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

berechnen. Dazu schreiben wir zunächst den Operator H_3 in der Form

$$H_3 \colon \begin{pmatrix} x \\ y \end{pmatrix} \to \underbrace{\begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{2} & 0 \end{pmatrix}}_{=:A_3} \begin{pmatrix} x \\ y \end{pmatrix} + \underbrace{\begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}}_{:=b_3} \qquad \text{also} \qquad H_3 \colon v \to A_3 v + b_3.$$

Damit erhalten wir, dass

$$\begin{aligned} H_3^2 v &= A_3(A_3 v + b_3) + b_3 = A_3^2 v + A_3 b_3 + b_3 \\ H_3^3 v &= A_3(A_3^2 v + A_3 b_3 + b_3) + b_3 = A_3^3 v + A_3^2 b + A_3 b_3 + b_3 \\ &\vdots \\ H_3^n v &= A_3^n v + A_3^{n-1} b_3 + \ldots + A_3 b_3 + b_3 \end{aligned}$$

Der Term $A_3^{n-1} b_3 + \ldots + A_3 b_3 + b_3$ ähnelt einer geometrischen Reihe. Wir wenden daher einen altbekannten Trick an:

$$\begin{aligned} (I - A_3)\left(A_3^{n-1} b_3 + \ldots + A_3 b_3 + b_3\right) &= A_3^{n-1} b_3 + \ldots + A_3 b_3 + b_3 \\ &\quad - A_3^n b_3 - A_3^{n-1} b_3 - \ldots - A_3 b_3 \\ &= b_3 - A_3^n b_3 = (I - A_3^n)\, b_3 \end{aligned}$$

Daraus folgt, dass

$$A_3^{n-1} b_3 + \ldots + A_3 b_3 + b_3 = (I - A_3)^{-1} (I - A_3^n)\, b_3.$$

Da $\lim_{n\to\infty} A_3^n = 0$ erhalten wir

$$\begin{aligned} \lim_{n\to\infty} H_3^n \begin{pmatrix} 0 \\ 0 \end{pmatrix} &= \lim_{n\to\infty} \left(A_3^n \begin{pmatrix} 0 \\ 0 \end{pmatrix} + A_3^{n-1} b_3 + \ldots + A_3 b_3 + b_3 \right) \\ &= \lim_{n\to\infty} \left((I - A_3)^{-1} (I - \underbrace{A_3^n}_{\to 0}) b_3 \right) = (I - A_3)^{-1} b_3 \end{aligned}$$

Das heißt, wir finden den Funktionswert $h(1) = h\,(0_4.3333\ldots) =: \begin{pmatrix} x \\ y \end{pmatrix}$ durch Lösen des Gleichungssystems

$$(I - A_3) \begin{pmatrix} x \\ y \end{pmatrix} = b_3 \qquad \Leftrightarrow \qquad \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}$$

Da die rechte Seite identisch zur ersten Spalte der Matrix des Gleichungssystems ist, können wir die Lösung $x = 1$ und $y = 0$ direkt ablesen. Damit gilt

$$h\,(0_4.3333\ldots) = h(1) = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

was auch unserer Erwartung entspricht – die Hilbertkurve endet in der rechten unteren Ecke des Einheitsquadrats.

Eindeutigkeit der Hilbertfunktion
Für die Eindeutigkeit der Hilbertfunktion bleibt uns nun zu zeigen, dass

$$H_{q_n}\begin{pmatrix}0\\0\end{pmatrix} = H_{q_n-1}\begin{pmatrix}1\\0\end{pmatrix} \qquad \text{für } n = 1,2,3.$$

Wir zeigen dies exemplarisch für $n = 1$:

$$\begin{aligned} H_1\begin{pmatrix}0\\0\end{pmatrix} &= \begin{pmatrix}\frac{1}{2} & 0\\ 0 & \frac{1}{2}\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0\\\frac{1}{2}\end{pmatrix} = \begin{pmatrix}0\\\frac{1}{2}\end{pmatrix} \qquad \text{und} \\ H_0\begin{pmatrix}1\\0\end{pmatrix} &= \begin{pmatrix}0 & \frac{1}{2}\\ \frac{1}{2} & 0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\\frac{1}{2}\end{pmatrix} \quad \text{q.e.d.} \end{aligned}$$

4.4 Berechnung der Umkehrfunktion

Bisher haben wir zu einem gegebenen Parameter t den zugehörigen Punkt $h(t)$ auf der Kurve bestimmt. Von ebenso großer praktischer Bedeutung ist jedoch das Umkehrproblem:

Finde zu einem gegebenen Punkt $(x, y) \in \mathcal{Q}$ einen Parameter t, so dass
$$h(t) = (x, y).$$

Für das Anwendungsbeispiel multidimensionaler Daten entspricht dies der Aufgabe, zu einem vorgegebenen Tupel (x, y) den Speicherplatz zu finden, an dem dieses Tupel angelegt ist.

Eindeutigkeit der Umkehrung
Zuerst erinnern wir uns an ein früheres Ergebnis: die Hilbertkurve (ebenso die Peanokurve) ist surjektiv, aber nicht bijektiv! Daher gibt es zu einem Punkt (x, y) unter Umständen mehrere geeignete Parameter t. Wir folgern:

- Eine Umkehrabbildung h^{-1} im strengen Sinn existiert nicht.
- Wir können lediglich eine technisch eindeutig gemachte Abbildung $\bar{h}^{-1}$ definieren und berechnen.
- Den Parameter $\bar{h}^{-1}(x, y)$ werden wir als *Hilbertindex* von (x, y) bezeichnen.

Die Berechnung des Hilbertindex erfolgt wieder rekursiv und entspricht gerade der Umkehrung des Algorithmus, der zu einem Parameter den zugehörigen Kurvenpunkt bestimmt:

1. Wir bestimmen das Teilquadrat, in dem (x, y) liegt

2. Mit Hilfe der Umkehrabbildungen der $H_0, \ldots, H_3$ bilden wir den Punkt (x, y) zurück ins Ausgangsquadrat ab. Dies liefert uns den Punkt $(\tilde{x}, \tilde{y})$.

3. Wir berechnen rekursiv den zu $(\tilde{x}, \tilde{y})$ gehörigen Parameter $\tilde{t}$

4. Abhängig vom oben bestimmten Teilquadrat berechnen wir aus $\tilde{t}$ den Hilbertindex t.

Bestimmung der Umkehrabbildungen zu $H_0, \ldots, H_3$
Die Umkehrabbildungen zu $H_0, \ldots, H_3$ erhalten wir durch Auflösen der jeweiligen Transformationen nach den Ausgangsvariablen. Für H_0 zum Beispiel erhalten wir:

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} = H_0 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2}y \\ \frac{1}{2}x \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2\tilde{y} \\ 2\tilde{x} \end{pmatrix}$$

Durch analoge Rechnung für H_1, H_2 und H_3 erhalten wir schließlich folgende Umkehrabbildungen:

$$H_0^{-1} := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} 2y \\ 2x \end{pmatrix} \qquad H_1^{-1} := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} 2x \\ 2y-1 \end{pmatrix}$$

$$H_2^{-1} := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} 2x-1 \\ 2y-1 \end{pmatrix} \quad H_3^{-1} := \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} -2y+1 \\ -2x+2 \end{pmatrix}$$

Algorithmus zur Berechnung der Umkehrfunktion
Die (technisch eindeutig gemachte) Umkehrabbildung $\bar{h}^{-1}$ erhalten wir nun, indem wir die Operationen der Berechnung der Hilbertabbildung schrittweise umdrehen:

(1) Zum gegebenen Punkt (x, y) ist zuerst abhängig von $x <> \frac{1}{2}$ und $y <> \frac{1}{2}$: das Teilquadrat zu bestimmen, in dem (x, y) liegt. Nach dem Schema

1	2
0	3

können wir die Nummer $q \in \{0, \ldots, 3\}$ dieses Teilquadrats bestimmen.

Die ambivalenten Fälle $x = \frac{1}{2}$ und/oder $y = \frac{1}{2}$ müssen dabei eindeutig entweder der Situation $<$ oder $>$ zugeordnet werden. Dies liefert die *Eindeutigkeit* der Abbildung.

(2) Mittels der richtigen Umkehrabbildung ermitteln wir als nächstes die relative Lage von (x, y) im Teilquadrat q, und zwar $(\tilde{x}, \tilde{y}) := H_q^{-1}(x, y)$

(3) Als nächstes wird (rekursiv!) der Hilbertindex $\tilde{t}$ des transformierten Punkts $(\tilde{x}, \tilde{y})$ im Teilquadrat q bestimmt: $\tilde{t} := \bar{h}^{-1}(\tilde{x}, \tilde{y})$

(4) der Hilbertindex von (x, y) ist dann $t := \frac{1}{4}(q + \tilde{t})$.

Zu diesem rekursiven Schema ist nur noch ein geeignetes Abbruchkriterium zu ergänzen. Wir geben dazu eine Toleranz ϵ für den gelieferten Index vor. Im Fall $\epsilon > 1$ dürfen wir jeden beliebigen Wert aus dem Intervall $[0, 1]$, dem Wertebereich des Hilbertindex, als Ergebnis liefern. Mit jedem rekursiven Aufruf dürfen wir diese Schranke um den Faktor 4 erhöhen – mit der Transformation eines Teilquadrats auf das Ausgangsquadrat wird nämlich auch das zugehörige Teilintervall um den Faktor 4 gestreckt. Damit erhalten wir die Funktion `hilbIndex` zur Berechnung des Hilbertindex.

4.5 Arithmetisierung der Peanoabbildung

Die Arithmetisierung der Peanoabbildung erfolgt völlig analog zu der der Hilbertkurve. Anstelle der Vierteilung von Intervallen und Teilquadraten wird nun jedoch eine Unterteilung in jeweils neuen Teile vorgenommen. Deswegen gehen wir im Folgenden von der Darstellung des Parameters t im „Nonalsystem“ aus, also $t = 0_9.n_1n_2n_3n_4\ldots$. Anschließend suchen wir

Funktion `hilbIndex`

```
func hilbIndex(x,y,eps) begin
    if eps > 1 then return 0;
    if  x < 0.5 then
        if y < 0.5 then
            return ( 0 + hilbIndex(2*y, 2*x, 4*eps) )/4;
        else
            return ( 1 + hilbIndex(2*x, 2*y - 1, 4*eps) )/4;
        end
    else
        if y ≥ 0.5 then
            return ( 2 + hilbIndex(1-2*y, 2-2*x, 4*eps) )/4;
        else
            return ( 3 + hilbIndex(2*x-1, 2*y - 1, 4*eps) )/4;
        end
    end
end func
```

Operatoren $P_0, \ldots, P_8$, so dass

$$p(0_9.n_1n_2n_3n_4\ldots) = P_{n_1} \circ P_{n_2} \circ P_{n_3} \circ P_{n_4} \circ \cdots \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Die Operatoren führen wir im Folgenden so auf, dass die Abfolge der neun Teilquadrate in der rekursiven Konstruktion der Peanokurve wiedergegeben wird:

$$P_2\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x + 0 \\ \frac{1}{3}y + \frac{2}{3} \end{pmatrix} \quad P_3\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x + \frac{1}{3} \\ -\frac{1}{3}y + 1 \end{pmatrix} \quad P_8\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x + \frac{2}{3} \\ \frac{1}{3}y + \frac{2}{3} \end{pmatrix}$$

$$P_1\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -\frac{1}{3}x + \frac{1}{3} \\ \frac{1}{3}y + \frac{1}{3} \end{pmatrix} \quad P_4\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -\frac{1}{3}x + \frac{2}{3} \\ -\frac{1}{3}y + \frac{2}{3} \end{pmatrix} \quad P_7\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -\frac{1}{3}x + 1 \\ \frac{1}{3}y + \frac{1}{3} \end{pmatrix}$$

$$P_0\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x + 0 \\ \frac{1}{3}y + 0 \end{pmatrix} \quad P_5\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x + \frac{1}{3} \\ -\frac{1}{3}y + \frac{1}{3} \end{pmatrix} \quad P_6\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{3}x + \frac{2}{3} \\ \frac{1}{3}y \end{pmatrix}$$

Die Operatoren skalieren den übergebenen Vektor nun mit dem Faktor $\frac{1}{3}$, da sich die Teilquadrate bei der Peanokurve durch Drittelung der Seitenlängen ergeben. Zu den Operatoren ist folgendes zu bemerken:

- bei der Peanokurve ist für keinen der Operatoren eine Rotation bzw. eine Vertauschung der x- und y-Koordinaten erforderlich. Die „Schlängelung“ der Kurve bleibt stets vertikal. Man kommt mit Spiegelungen aus.
- die Operatoren P_0, P_2, P_6, und P_8 erhalten die Orientierung der Kurve – hier genügt die Skalierung und Verschiebung auf den neuen Startpunkt.
- die Operatoren P_1 und P_7 erfordern eine Spiegelung an der vertikalen Achse, d.h. das Vorzeichen der x-Komponente muss sich ändern.
- die Operatoren P_3 und P_5 erfordern dagegen eine Spiegelung an der horizontalen Achse – entsprechend ist das Vorzeichen der y-Koordinate zu wechseln.
- der Operator P_4 spiegelt an x- und y-Achse.

Analog zum Vorgehen bei der Hilbertkurve lässt sich die Arithmetisierung wieder in eine Funktion zur Berechnung der Peanoabbildung bzw. des Peanoindex umsetzen – siehe die Funktion `peanoIndex`.

4.6 Definition der Peanokurve nach Peano

Giuseppe Peano verwendete eine etwas andere Darstellung seiner Peanokurve als die, die wir in den vorhergehenden Abschnitten vorgestellt haben. Seine Darstellung beruht auf der Darstellung des Parameters als Ternärzahl.

Definition 4.1 (Peanokurve, Originalkonstruktion nach Peano)
Sei ein Parameter $t \in \mathcal{I} := [0,1]$ in Ternärdarstellung gegeben:

$$t = 0_3.t_1t_2t_3t_4\ldots$$

Dann ist die Peano-Abbildung $p\colon \mathcal{I} \to \mathcal{Q} := [0,1] \times [0,1]$ definiert gemäß

$$p(t) := \begin{pmatrix} 0_3.t_1\, k^{t_2}(t_3)\, k^{t_2+t_4}(t_5)\ldots \\ 0_3.k^{t_1}(t_2)\, k^{t_1+t_3}(t_4)\ldots \end{pmatrix}$$

wobei der Operator k definiert ist als $k(t_i) := 2 - t_i$ für $t_i = 0,1,2$, und k^j die j-fache Verkettung der Funktion k bedeutet.

Funktion `peanoIndex`

```
func peanoIndex begin
    if tiefe = 0 then
        return (0,0)
    else
        // nächste Quarternärstelle bestimmen
        q := floor(9*t);
        r := 9*t - q;
        (x,y) := peanoIndex(r,tiefe-1);
        switch q do
            case 0: return ( x/3, y/3 ) ;
            case 1: return ( (-x+1)/3, (y+1)/3 ) ;
            case 2: return ( x/3, (y+2)/3 ) ;
            case 3: return ( (x+1)/3, (-y+3)/3 ) ;
            case 4: return ( (-x+2)/3, (-y+2)/3 ) ;
            case 5: return ( (x+1)/3, (-y+1)/3 ) ;
            case 6: return ( (x+2)/3, y/3 ) ;
            case 7: return ( (-x+3)/3, (y+1)/3 ) ;
            case 8: return ( (x+2)/3, (y+2)/3 ) ;
        end
    end
end func
```

Zu beachten ist, dass die Ternärentwicklung nicht nach endlich vielen Stellen abgebrochen werden darf, da man sonst in der Regel ein falsches Ergebnis erhält:

$$\begin{aligned}
p(0_3.11) &\stackrel{?}{=} \begin{pmatrix} 0_3.1 \\ 0_3.k^1(1) \end{pmatrix} = \begin{pmatrix} 0_3.1 \\ 0_3.1 \end{pmatrix} = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \end{pmatrix} \\
p(0_3.1100) &\stackrel{?}{=} \begin{pmatrix} 0_3.1k^1(0) \\ 0_3.k^1(1)k^1(0) \end{pmatrix} = \begin{pmatrix} 0_3.12 \\ 0_3.12 \end{pmatrix} = \begin{pmatrix} \frac{5}{9} \\ \frac{5}{9} \end{pmatrix} \\
p(0_3.110000\ldots) &\stackrel{!}{=} \begin{pmatrix} 0_3.1k^1(0)k^1(0)\ldots \\ 0_3.k^1(1)k^1(0)k^1(0)\ldots \end{pmatrix} = \begin{pmatrix} 0_3.122\ldots \\ 0_3.122\ldots \end{pmatrix} \\
&= \begin{pmatrix} 0_3.2 \\ 0_3.2 \end{pmatrix} = \begin{pmatrix} \frac{2}{3} \\ \frac{2}{3} \end{pmatrix}.
\end{aligned}$$

Es muss also stets die „unendliche“ Ternärentwiklung verwendet werden.

Zum besseren Verständnis können wir zudem einige Eigenschaften des Operators k festhalten:

$$k(1) = 1 \quad \Rightarrow \quad k^n(1) = 1 \tag{4.3}$$

$$k^2(t) = t \quad \Rightarrow \quad k^{2n}(t) = t \quad \text{für alle } n. \tag{4.4}$$

Der Operator k ist für die passende Umkehr der Laufrichtung in den Schlängelungen der Peanokurve verantwortlich.

Gleichwertigkeit der Darstellung von Peano mit der bisherigen Arithmetisierung

Als nächstes müssten wir zeigen, dass Definition 4.1 tatsächlich eine raumfüllende Kurve definiert, also stetig und surjektiv ist. Dies weisen wir nach, indem wir zeigen, dass die in der Definition 4.1 festgelegte Abbildung identisch ist zu der bisher verwendeten. Dazu zeigen wir induktiv, dass die Arithmetisierung der Peanokurve die gleichen Funktionswerte liefert wie obige Defintion.

Zunächst stellen wir fest, dass jeweils zwei Stellen der Ternärentwicklung eines Parameters t einer Stelle seiner Entwicklung im Nonalsystem entsprechen. Wenn also

$$0_9.n_1n_2\cdots = t = 0_3.t_1t_2t_3t_4\ldots,$$

dann gilt

$$\begin{aligned} n_1 &= 3 * t_1 + t_2, \\ n_2 &= 3 * t_3 + t_4, \\ &\vdots \\ n_i &= 3 * t_{2i-1} + t_{2i}, \\ &\vdots \end{aligned}$$

Damit bleibt zu zeigen, dass

$$\begin{aligned} p(t) &= \begin{pmatrix} 0_3.t_1\, k^{t_2}(t_3)\, k^{t_2+t_4}(t_5)\dots \\ 0_3.k^{t_1}(t_2)\, k^{t_1+t_3}(t_4)\dots \end{pmatrix} \\ &= P_{3t_1+t_2} \circ P_{3t_3+t_4} \circ \dots \circ P_{3t_{2n-1}+t_{2n}} \circ \dots \circ \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Dies geschieht durch vollständige Induktion über die Stellen n_i. So gilt etwa für $t = 0_3.11$

$$P_{3+1} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{1}{3} & 0 \\ 0 & -\frac{1}{3} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} \frac{2}{3} \\ \frac{2}{3} \end{pmatrix} = \begin{pmatrix} \frac{2}{3} \\ \frac{2}{3} \end{pmatrix}$$

und

$$\begin{pmatrix} 0_3.1k^1(0)k^1(0)\dots \\ 0_3.k^1(1)k^1(0)k^1(0)\dots \end{pmatrix} = \begin{pmatrix} 0_3.122\dots \\ 0_3.122\dots \end{pmatrix} = \begin{pmatrix} 0_3.2 \\ 0_3.2 \end{pmatrix} = \begin{pmatrix} \frac{2}{3} \\ \frac{2}{3} \end{pmatrix}$$

Hier ist wieder zu beachten, dass die Ternärentwicklung nicht nach 2 Stellen abgebrochen werden darf! In gleicher Weise können wir nun die restlichen acht Fälle ($t_1, t_2 \in \{0, 1, 2\}$) untersuchen und damit den Induktionsstart beweisen.

In gleicher Weise kann auch der Induktionsschluss in neun Fälle unterschieden werden. Dabei kommen in jedem Induktionsschritt die beiden höchstwertigen Ternärstellen, t_1 und t_2, neu hinzu. Als Beispiel betrachten wir den Fall

$$t = 0_3.21t_3t_4\dots t_{2n-1}.$$

Es ist zu zeigen, dass

$$\begin{aligned} p(t) &= P_{3\cdot 2+1} \circ P_{3t_3+t_4} \circ \ldots \circ P_{3t_{2n-1}+t_{2n}} \circ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0_3.2\ k^1(t_3) \ldots k^{1+t_4+\cdots+t_{2n-2}}(t_{2n-1}) \ldots \\ 0_3.k^2(1)\ k^{2+t_3}(t_4) \ldots k^{2+t_3+\cdots+t_{2n-1}}(t_{2n}) \ldots \end{pmatrix} \\ &= \begin{pmatrix} 0_3.2\ k(t_3) \ldots k(k^{t_4+\cdots+t_{2n-2}}(t_{2n-1})) \ldots \\ 0_3.1\ k^{t_3}(t_4) \ldots k^{t_3+\cdots+t_{2n-1}}(t_{2n}) \ldots \end{pmatrix} \end{aligned} \tag{4.5}$$

falls als Induktionsvoraussetzung gilt, dass die beiden Darstellungen für den Parameter $\tilde{t} = 0_3.t_3t_4 \ldots t_{2n-1}t_{2n}$ übereinstimmen, also

$$\begin{aligned} p(\tilde{t}) &= P_{3t_3+t_4} \circ \ldots \circ P_{3t_{2n-1}+t_{2n}} \circ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0_3.k^1(t_3)\ \ldots k^{1+t_4+\cdots+t_{2n-2}}(t_{2n-1}) \ldots \\ 0_3.k^{t_3}(t_4) \ldots k^{2+t_3+\cdots+t_{2n-1}}(t_{2n}) \ldots \end{pmatrix}. \end{aligned}$$

Noch einmal erinnern wir uns: im Falle eines endlichen Nonalbruches dürfen wir die reguläre Arithmetisierung abbrechen, sobald nur noch Nullstellen folgen. In der ursprünglichen Peano-Formulierung müssen wir jedoch immer einen unendlichen Ternärbruch berücksichtigen.

Führen wir den Induktionsschluss fort: mit Induktionsvoraussetzung gilt nun

$$\begin{aligned} p(t) &= P_7 \circ P_{3t_3+t_4} \circ \ldots \circ P_{3t_{2n-1}+t_{2n}} \circ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ &= P_7 \circ \begin{pmatrix} 0_3.k(t_3)\ k^{1+t_4}(t_5) \ldots k^{1+t_4+\cdots+t_{2n-2}}(t_{2n-1}) \ldots \\ 0_3.k^{t_3}(t_4) \ldots k^{2+t_3+\cdots+t_{2n-1}}(t_{2n}) \ldots \end{pmatrix} \end{aligned}$$

Zur besseren Übersichtlichkeit verwenden wir im Folgenden die Kurzbezeichnungen $s_g := t_4 + \cdots + t_{2n-2}$ sowie $s_u := t_3 + \cdots + t_{2n-1}$. Wir rechnen weiter und erhalten

$$\begin{aligned} p(t) &= \begin{pmatrix} -\frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 0_3.k(t_3)\ k^{1+t_4}(t_5) \ldots k^{1+s_g}(t_{2n-1})\ k^{1+s_g}(0) \ldots \\ 0_3.k^{t_3}(t_4) \ldots k^{2+s_u}(t_{2n})\ k^{2+s_u}(0) \ldots \end{pmatrix} + \begin{pmatrix} 1 \\ \frac{1}{3} \end{pmatrix} \\ &= \begin{pmatrix} 1 - 0_3.0\ k(t_3)\ k^{1+t_4}(t_5) \ldots k^{1+s_g}(t_{2n-1})\ k^{1+s_g}(0) \ldots \\ 0_3.1\ k^{t_3}(t_4) \ldots k^{2+s_u}(t_{2n})\ k^{2+s_u}(0) \ldots \end{pmatrix} \\ &= \begin{pmatrix} 0_3.222 \ldots - 0_3.0\ k(t_3)\ k^{1+t_4}(t_5) \ldots k^{1+s_g}(t_{2n-1})\ k^{1+s_g}(0) \ldots \\ 0_3.1\ k^{t_3}(t_4) \ldots k^2(k^{s_u}(t_{2n}))\ k^2(k^{s_u}(0)) \ldots \end{pmatrix} \end{aligned}$$

Nun können wir in der ersten Komponente Stelle für Stelle subtrahieren. Dabei entspricht das Subtrahieren von der 2 gerade der Wirkung des Operators k, also zum Beispiel $2 - k(t_3) = k(k(t_3))$. Wir erhalten demnach

$$\begin{aligned} p(t) &= \begin{pmatrix} 0_3.2\ k(k(t_3))\ k(k^{1+t_4}(t_5)) \dots k(k^{1+s_g}(t_{2n-2}))\ k(k^{1+s_g}(0)) \\ 0_3.1\ k^{t_3}(t_4) \dots k^{s_u}(t_{2n})\ k^{s_u}(0) \dots \end{pmatrix} \\ &= \begin{pmatrix} 0_3.2\ t_3\ k^{t_4}(t_5) \dots k^{s_g}(t_{2n-2})\ k^{s_g}(0) \\ 0_3.1\ k^{t_3}(t_4) \dots k^{s_u}(t_{2n})\ k^{s_u}(0) \dots \end{pmatrix}, \end{aligned}$$

da $k(k(t)) = t$. Dies vergleichen wir mit der Behauptung (4.5), die wir für den Induktionsschluss zeigen wollten. Wir stellen fest, dass die Behauptung nach erheblicher Rechnerei bewiesen ist, und wir somit die Gleichheit der beiden Darstellungen der Peanokurve nachgewiesen haben.

5 Approximierende Polygone

5.1 Approximierende Polygone der Hilbert- und der Peanokurve

Bereits in Kapitel 2.3.2 haben wir die approximierenden Polygone der Hilbertkurve als Hilfsmittel zur Konstruktion der Hilbertkurve eingeführt. Mit Hilfe der Arithmetisierung der Hilbertkurve können wir nun eine mathematische Definition der approximierenden Polygone nachliefern.

Definition 5.1 (approximierendes Polygon der Hilbertkurve)
Der Streckenzug, der die $4^n + 1$ *Punkte*

$$h(0), h(1 \cdot 4^{-n}), h(2 \cdot 4^{-n}), \ldots, h((4^n - 1) \cdot 4^{-n}), h(1)$$

verbindet, heißt n*-tes* approximierendes Polygon der Hilbertkurve.

In Abbildung 5.1 sind noch einmal die ersten drei approximierenden Polygone der Hilbertkurve abgebildet.

Eigenschaften approximierender Polygone
Zunächst wiederholen wir einige der Eigenschaften der approximierenden Polygone:

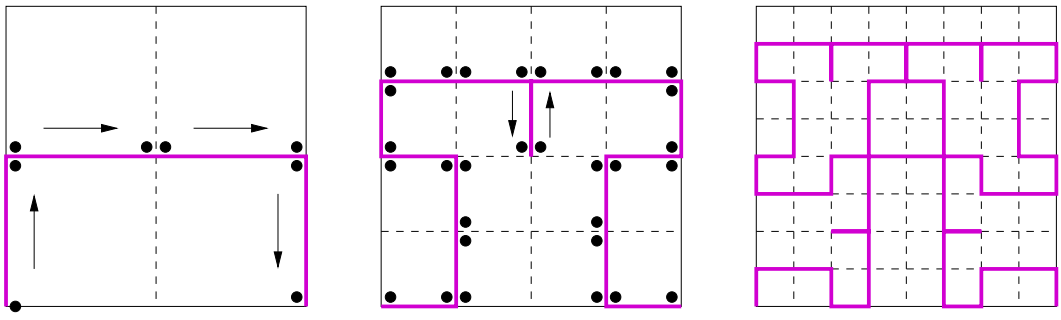


Abbildung 5.1: Die ersten drei approximierenden Polygone der Hilbertkurve.

- Die approximierenden Polygone verbinden die *Eckpunkte* der rekursiv unterteilten Quadrate.
- Die verbundenen Eckpunkte sind Start- und Endpunkte der Hilbertkurve im entsprechenden Teilquadrat. Die Polygone sind auf diese Weise eine Hilfestellung zur Konstruktion der Hilbertkurve.

Die Arithmetisierung der Hilbertkurve und die resultierende Definition der approximierenden Polygone liefern uns nunmehr den Nachweis, dass die Eckpunkte der Polygone tatsächlich auf der Hilbertkurve liegen. Sie sind als Bilder der Intervallgrenzen gerade die „Übergangspunkte“ zwischen den Teilquadraten.

Zu den approximierenden Polygonen lassen sich natürlich eigene Abbildungsvorschriften $p_n(t)$ definieren. Als Bemerkung sei noch erwähnt, dass sich zeigen lässt, dass die Folge dieser Abbildungen $p_n(t)$ *gleichmäßig* gegen die Hilbertkurve konvergiert. Dies liefert unter anderem einen weiteren Beweis für die Stetigkeit der Hilbertkurve.

Approximierende Polygone der Peanokurve
Analog zur Hilbertkurve definieren wir auch für die Peanokurve die approximierenden Polygone.

Definition 5.2 (approximierendes Polygon der Peanokurve)
Der Streckenzug, der die $9^n + 1$ *Punkte*

$$p(0), p(1 \cdot 9^{-n}), p(2 \cdot 9^{-n}), \ldots, p((9^n - 1) \cdot 9^{-n}), p(1)$$

verbindet, heißt n-tes approximierendes Polygon der Peanokurve.

Die ersten beiden approximierenden Polygone sind in Abbildung 5.2 dargestellt. Wir erkennen das für die Peanokurven typische diagonale Durchlaufen der Teilquadrate.

Erzeugung über rekursive Wiederholung eines Leitmotivs
Wenn wir die sukzessive Bildung der approximierenden Polygone der Hilbertkurve und der Peanokurve untersuchen, erkennen wir ein wichtiges Bauprinzip der approximierenden Polygone. Wir sehen, dass wir das $(n + 1)$-te approximierende Polygon jeweils dadurch erhalten, dass wir jede Kante des n-ten Polygons durch das Grundpolygon ersetzen. Dabei müssen wir auf die korrekte Orientierung achten. Bei der Hilbertkurve muss

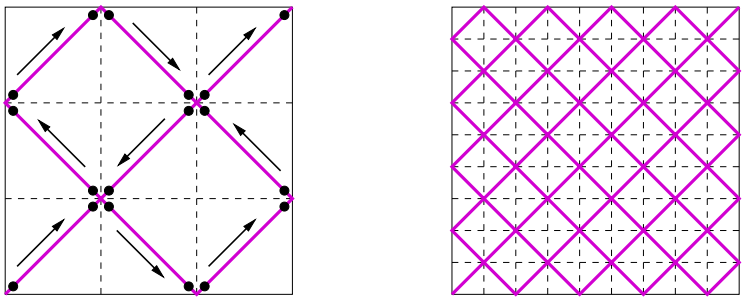


Abbildung 5.2: Die ersten beiden approximierenden Polygone der Peanokurve

das Grundpolygon stets innerhalb des zu der Kante gehörigen Teilquadrats liegen.

Die approxmierenden Polygone werden also durch rekursives Ersetzen der Kanten durch das Grundpolygon erzeugt – das Grundpolygon werden wir deshalb im Weiteren als *Leitmotiv* bezeichnen. Die daraus folgende Ähnlichkeit zu Kochkurven und Fraktalen werden wir im weiteren Verlauf dieses Kapitels näher untersuchen.

5.2 Approximierende Polygone als Längenmessung

Die approximierenden Polygone sind festgelegt als Streckenzug, der die Punkte

$$h(0), h(1 \cdot 4^{-n}), h(2 \cdot 4^{-n}), \ldots, h((4^n - 1) \cdot 4^{-n}), h(1)$$

miteinander verbindet. Die Eckpunkte des Polygons liegen sämtlich auf der Hilbertkurve. Dies entspricht gerade dem klassischen Vorgehen zur Messung der Länge einer Kurve: wir approximieren die zu messende Kurve, indem wir in regelmäßigen Abständen Punkte auf der Kurve herausgreifen, jeweils die Abstände zwischen den aufeinander folgenden Punkten berechnen und schließlich alle Einzelabstände aufsummieren.

Wenn wir die Abstände der Messpunkte immer kleiner machen, erwarten wir, dass die Messung immer genauer wird. Die Länge der zu messenden Kurve können wir so als Grenzwert der Längen der Streckenzüge definieren.

Wie lang ist die Hilbertkurve?
Wir können also die Länge der approximierenden Polygone berechnen, und diese als Schätzung der Länge der Hilbertkurve verwenden. Dabei beobachten wir folgendes:

- Die approximierenden Polygone werden durch rekursive Wiederholung des Leitmotivs gebildet. Ein zwei Einheiten langes Teilstück (eine Kante eines Teilquadrats) wird durch das vier Einheiten lange Leitmotiv ersetzt.
- Daher **verdoppelt** sich die Länge der approximierenden Polygone in jedem Rekursionsschritt.

Die Länge des 0-ten approximierenden Polygons ist 1, also ist die Länge des n-ten Polygons gleich 2^n. Für $n \to \infty$ wird diese Länge beliebig groß. Unsere Methode zur Längenmessung versagt hier, weil bei immer feinerer Auflösung eben kein Grenzwert existiert. Was können wir daraus folgern?

1. Eine „Länge“ der Hilbertkurve ist offenbar nicht vernünftig definierbar.
2. Stattdessen können wir der Hilbertkurve bequem eine „Fläche“ zuordnen, nämlich die Fläche des von ihr ausgefüllten Einheitsquadrats (also 1).

Dies wirft zwangsläufig die Frage auf, welche Dimension die Hilbertkurve eigentlich hat!

5.3 Längenmessung fraktaler Kurven

Kurven, die durch geeignetes, rekursives Anwenden eines Leitmotivs entstehen, sind eine wichtige Unterklasse der sogenannten *fraktalen* Kurven. Die „Längenmessung“ der Hilbertkurve können wir analog auf jede derartige fraktale Kurve anwenden:

- Üblicherweise wird die (unendliche) fraktale Kurve durch endliche Iterationen approximiert, wobei die Iterationen meist Streckenzüge sind. Die Eckpunkte der Streckenzüge liegen immer dann direkt auf der fraktalen Kurven, wenn die Endpunkte der Strecken auch stets

Endpunkte des Leitmotivs bleiben. Haben die einzelnen Strecken des Streckenzugs alle die Länge ϵ, dann charakterisiert ϵ die Genauigkeit der Längenmessung. Wir können also $L(\epsilon)$, d.h. die abhängig von der Genauigkeit ϵ gemessene Länge der fraktalen Kurve, als Länge des entsprechenden endlichen Streckenzugs berechnen.

- Aufgrund der rekursiven Wiederholung des Leitmotivs erhalten wir eine einfache rekursive Rechenvorschrift für $L(\epsilon)$. Wird bei einer fraktalen Kurve in jeder Iteration jede r Einheiten lange Strecke durch ein q Einheiten langes Leitmotiv ersetzt, dann erhöht sich die Länge der Iterationen jeweils um den Faktor $\frac{q}{r}$. Da sich die Messgenauigkeit dabei um den Faktor $\frac{1}{r}$ reduziert, erhalten wir folgende rekursive Formel für die Länge der Iterationen.

$$L\left(\frac{\epsilon}{r}\right) = \frac{q}{r} L(\epsilon), \qquad L(1) := \lambda$$

- Durch wiederholtes Anwenden dieser Längenformel erhalten wir, dass

$$L\left(\frac{1}{r^n}\right) = \left(\frac{q}{r}\right)^n L(1) = \frac{q^n}{r^n}\lambda$$

Setzen wir $\epsilon := \frac{1}{r^n}$ erhalten wir daraus

$$L(\epsilon) = \frac{q^n}{r^n}\lambda = \epsilon q^n \lambda \qquad \text{wobei} \quad r^n = \frac{1}{\epsilon} \quad \text{bzw.} \quad n = -\log_r \epsilon$$

Damit können wir wie folgt umformen:

$$L(\epsilon) = \lambda \epsilon q^{-\log_r \epsilon} = \lambda \epsilon \epsilon^{-\log_r \epsilon \cdot \log_\epsilon q} = \lambda \epsilon^{1-D}, \quad \text{wobei} \quad D = \frac{\log q}{\log r}$$

Die Längenmessung einer rekursiv definierten fraktalen Kurve liefert also folgende Formel:

$$L(\epsilon) = \lambda \epsilon^{1-D} \qquad \text{mit} \quad D = \frac{\log q}{\log r} \tag{5.1}$$

Nur im Fall $D = 1$ erhalten wir demnach eine vernünftige Länge. Der Fall $D = 1$ ist aber leider besonders uninteressant, da hier $q = r$, also eine r Einheiten lange Strecke jeweils durch ein $q = r$ Einheiten langes Leitmotiv

ersetzt wird – was demnach auch wieder eine Strecke sein muss (alle anderen Verbindungen wären ja länger). Die Längenmessung der entstehenden geraden Strecke ist allerdings ein recht langweiliges Problem.

Etwas interessanter wird es, wenn wir uns zum Vergleich die Längenmessung „gewöhnlicher“ Kurven ansehen, z.B. die des Kreisumfangs. Hier wissen wir, dass mit immer kleinerer Messgenauigkeit ϵ die Länge $K(\epsilon)$ gegen einen festen Wert konvergiert, nämlich π mal dem Kreisdurchmesser. Zumindest asymptotisch bekommen wir hier ein Verhalten gemäß der Formel $K(\epsilon) = \lambda\epsilon^0$. Dies legt uns nahe, dass λ in Gleichung (5.1) so etwas wie die Länge festlegt. Für was steht dann der Wert D?

Dass D für eine Strecke, also für das typische eindimensionale Objekt, gerade $D = 1$ ist, gibt einen Hinweis darauf, dass D eine Dimension sein könnte. Wie sieht das für die zweidimensionalen Kurven aus? Für die Hilbertkurve ist $r = 2$ und $q = 4$, für die Peanokurve $r = 3$ und $q = 9$. Wir erhalten die Werte

$$D = \frac{\log 4}{\log 2} = 2 \qquad \text{bzw.} \qquad D = \frac{\log 9}{\log 3} = 2.$$

In der Tat zeigt sich also, dass sowohl die zweidimensionale Hilbertkurve als auch alle zweidimensionalen Peanokurven gerade zu einer Längenformel mit $D = 2$ führen. Nachdem diese Kurven eine Fläche, also ein zweidimensionales Objekt, ausfüllen, scheint D tatsächlich so etwas wie die „wahre Dimension“ einer fraktalen Kurve anzugeben. Wir halten fest:

- D ist die *fraktale Dimension* der Kurve.
- λ ist die Länge bzgl. dieser Dimension.

Werfen wir einen vorsichtigen Blick in das Kapitel 6 über dreidimensionale raumfüllende Kurven. Dort erkennt man zum Beispiel in Abbildung 6.4, dass für die approximierenden Polygone der dort konstruierten 3D-Hilbertkurve $r = 2$ und $q = 8$ gilt. Dies führt gerade zum Wert $D = 3$.

Gleichung (5.1) liefert also in Gestalt der Konstante D einen für fraktale und raumfüllende Kurven „vernünftigen“ Dimensionsbegriff:

- Die fraktale Dimension der Hilbertkurve stimmt mit der geometrischen Dimension des von ihr ausgefüllten Gebiets überein. Gleiches gilt für die Peanokurve.

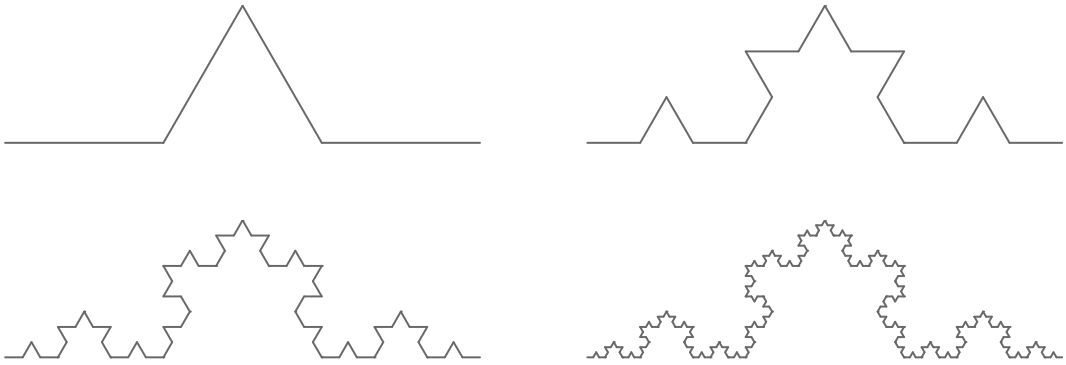


Abbildung 5.3: Die ersten vier Iterationen der Kochkurve

- Misst man die Länge der beiden Kurven in irgendeiner anderen Dimension, so erhält man entweder 0 oder ∞. So ist das Volumen des von der Hilbertkurve ausgefüllten Einheitsquadrates 0, wenn man es im Dreidimensionalen misst. Ebenso ergab sich die „herkömmliche“ Länge der (geometrisch) zweidimensionalen Hilbertkurve als unendlich.

Der entsprechende Dimensionsbegriff ist allgemein als *Hausdorff*-Dimension bekannt.

5.4 Exkurs: Fraktale Kurven

Die Bildung der approximierenden Polygone über die rekursive Wiederholung eines Leitmotivs ist identisch mit der Konstruktion Fraktaler Kurven vom Typ der bekannten *Kochkurve*. Das Leitmotiv der Kochkurve wird gebildet, indem man aus einer Strecke das mittlere Drittel entfernt und durch die beiden Schenkel eines gleichseitigen Dreiecks ersetzt. Rekursiv wird dann in jeder weiteren Iteration der Kochkurve jede Teilstrecke durch dieses Leitmotiv ersetzt. Die ersten vier der so entstehenden Iterationen sind in Abbildung 5.3 dargestellt.

Die Kochkurve ist insofern ein interessantes Studienobjekt, als sie tatsächlich eine Kurve – also stetig – ist, aber an keiner Stelle eine Steigung der Kurve angegeben werden könnte. Sie ist nirgends differenzierbar.

Von besonderem Interesse ist sie jedoch im Zusammenhang mit der Frage der Längenmessung und der Dimensionsbestimmung, so wie wir sie

im vorherigen Abschnitt eingeführt haben. Das Leitmotiv ersetzt in jedem Interationsschritt eine 3 Einheiten lange Strecke durch einen 4 Einheiten langen Streckenzug. Gemäß Gleichung 5.1 erhalten wir also für die Länge der auf der Einheitsstrecke gebildeten Kochkurve die Formel

$$L(\epsilon) = \epsilon^{1-\frac{\log 4}{\log 3}} \approx \epsilon^{1-1.26186}, \tag{5.2}$$

wenn ϵ die Länge der Streckenstücke in der gewählten Iteration ist. Ebenso erhalten wir, dass die Hausdorff-Dimension der Kochkurve

$$D = \frac{\log 4}{\log 3} \approx 1.26186$$

ist. Dies trägt der Tatsache Rechnung, dass die Kochkurve weder eine normale Kurve ist, noch eine richtige Fläche. Die Längenmessung der Kurve scheitert, weil die Länge mit immer feinerer Auflösung gegen unendlich strebt. Die Fläche der Kochkurve ist dagegen 0.

Diese „gebrochene“ Dimension hat für Kurven wie der Kochkurve den Begriff *fraktale Kurven* geprägt. Benoit Mandelbrot[1] hat die Verwendung fraktaler Kurven zur Beschreibung quasi-natürlicher Objekte begründet (siehe zum Beispiel die Koch'sche Schneeflocke in Abbildung 5.4). Der folgende Abschnitt stellt sein berühmtes Beispiel zur Längenmessung von Küstenlinien vor.

Wie lang ist die Küste Britanniens?

Wir stellen uns vor, dass wir die Länge der Küste der britannischen Insel bestimmen müssen. Für eine erste grobe Schätzung könnten wir zum Beispiel einen Atlas zu Hilfe nehmen. Wir stellen einen Stechzirkel auf die Länge 1 cm ein und stochern uns am Küstenrand entlang, bis wir einmal um die Küste herumgegangen sind. Die Anzahl der Zentimeterstücke zählen wir mit und erhalten mit Hilfe des Kartenmaßstabs eine Schätzung der Küstenlänge. Natürlich ist diese Schätzung noch sehr ungenau.

Daher können wir uns als nächstes einen Satz topographischer Karten mit viel kleinerem Maßstab besorgen und unsere Messung wiederholen. Da der genauere Maßstab viel mehr Details der Küste wiedergibt – kleinere Buchten werden sichtbar, etc. –, erhalten wir vermutlich einen deutlich

[1] Für Interessierte sei an dieser Stelle Mandelbrots Buch „Die fraktale Geometrie der Natur“ wärmstens zum Lesen empfohlen!

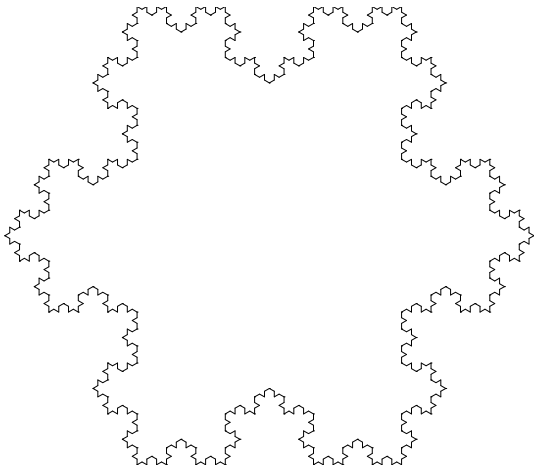


Abbildung 5.4: Die Koch'sche Schneeflocke

größeren Wert für die Länge der Küste. Abbildung 5.5 zeigt diesen Effekt exemplarisch, indem die Messung der Küstenlänge für zwei verschiedene „Stechzirkel-Längen" durchgeführt wird. Bei der Messung mit der roten Linie wurde der Messabstand auf ein Drittel verkleinert. Dadurch wächst die gemessene Küstenlänge von 108 auf 149 Einheiten.

In nächster Konsequenz könnten wir uns eine Vermessungsausrüstung besorgen, die etwa auf optischem Weg Längen zwischen zwei Meßstationen ermitteln kann. Wir könnten in einer Expedition entlang der gesamten Küste selbst die kleinsten Buchten berücksichtigen – wenn wir wollen, sogar einzelne Felsklippen und ähnliche Vorsprünge. Die Länge der Kurve würde mit wachsender Genauigkeit immer weiter ansteigen. Die Frage ist nun, ob diese Länge irgendwann gegen einen Grenzwert strebt, wenn wir etwa zum Zollstock greifen und schließlich auch noch den kleinsten Kieselstein für die Längenmessung berücksichtigen.

Natürlich haben wir hier die Schwelle zum Gedankenexperiment längst überschritten. Dennoch spielt die Wahl der Auflösung bei der Längenmessung eine so große Rolle, dass – wie Mandelbrot erwähnt – die in verschiedenen Lexika angegebene Länge von Küsten und Grenzen um bis zu 20 % variieren kann. Darüber hinaus zeigt sich, dass die gemessenen Längen der Küsten in der Tat einem Gesetz wie in Gleichung (5.1) gehorchen. Wir müssen uns also fragen ob es in Wahrheit die fraktalen Kurven mit „ge-

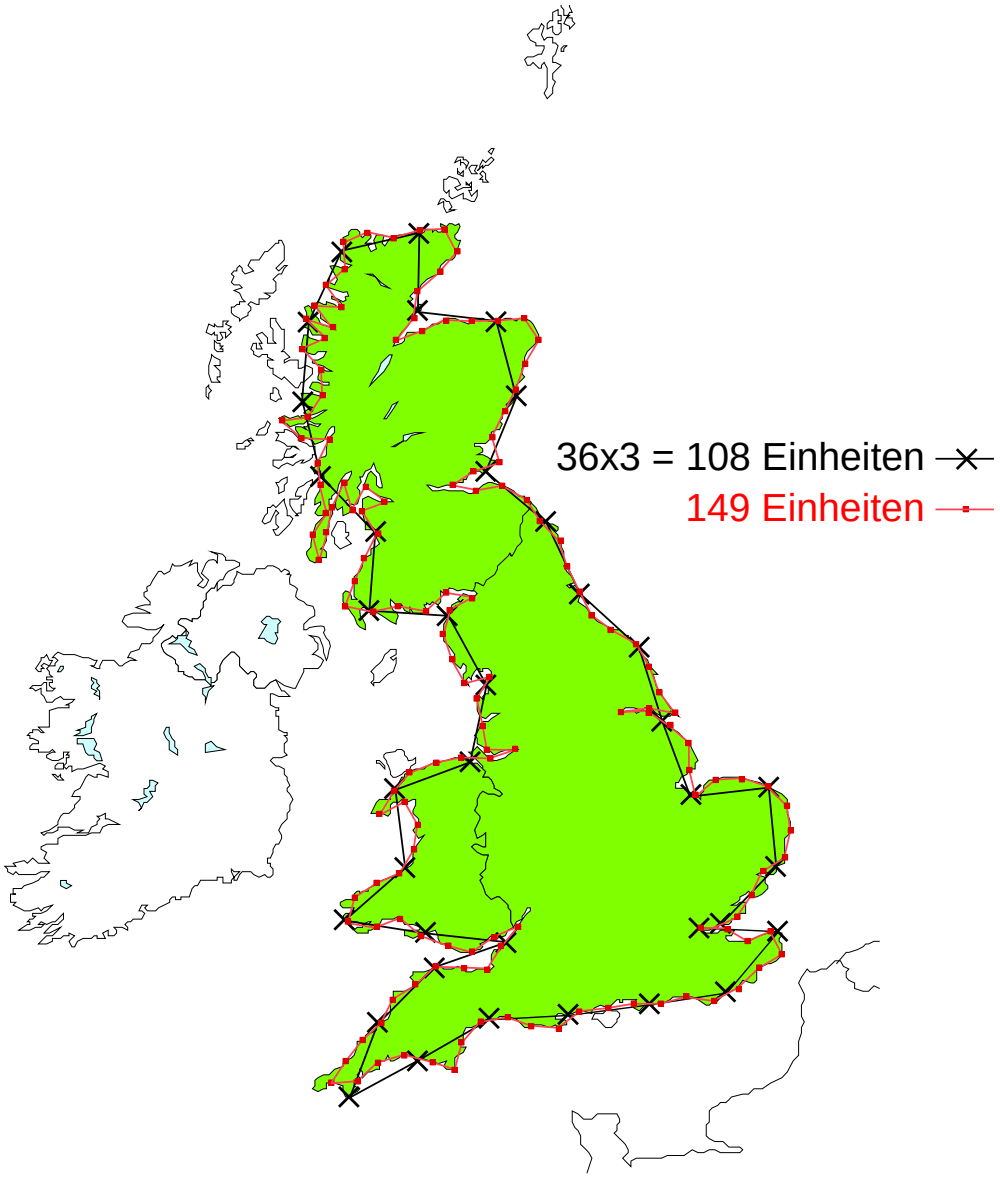


Abbildung 5.5: Längenmessung der Küste Britanniens – die Küstenlinie wird durch zwei Streckenzüge unterschiedlicher Genauigkeit approximiert.

brochener“ Dimension sind, die in der Natur vorkommen, und ob nicht die mathematisch exakten Kreisbögen und Geraden die eigentlichen mathematischen „Monster“ sind.

6 Dreidimensionale Raumfüllende Kurven

6.1 Charakterisierung raumfüllender Kurven

Bei der Konstruktion der Hilbertkurve und der verschiedenen Peanokurven haben wir uns beständig an zwei wesentliche Konstruktionsprinzipien gehalten:

1. Ausgehend von einer regelmäßigen Intervallschachtelung wurden jeweils Teilintervalle auf gleich große Teilquadrate abgebildet.
2. Die zu zwei benachbarten Teilintervallen gehörigen Teilquadrate waren stets wieder benachbart, d.h. sie besaßen mindestens eine gemeinsame Kante.

Wir werden deshalb für weitere raumfüllende Kurven, die ebenfalls nach diesen Prinzipien konstruiert werden, geeignete Bezeichnungen festlegen.

Definition 6.1 (*rekursive* raumfüllende Kurve)
Eine raumfüllende Kurve $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^d$ *heißt* rekursiv, *wenn sowohl* $\mathcal{I}$ *als auch* $\mathcal{Q}$ *in* m *Teilintervalle bzw. Teilgebiete zerlegt werden kann, so dass*

- $f_*(\mathcal{I}^{(\mu)}) = \mathcal{Q}^{(\mu)}$ *für alle* $\mu = 1, \ldots, m$, *und*
- *die* $\mathcal{Q}^{(\mu)}$ *geometrisch ähnlich zu* $\mathcal{Q}$ *sind.*

Definition 6.2 (*geschlossene* raumfüllende Kurve)
Eine rekursive raumfüllende Kurve heißt geschlossen, *wenn zu zwei benachbarten Intervallen* $\mathcal{I}^{(\nu)}$ *und* $\mathcal{I}^{(\mu)}$ *auch die zugehörigen Teilgebiete* $\mathcal{Q}^{(\nu)}$ *und* $\mathcal{Q}^{(\mu)}$ *direkt benachbart sind, d.h. mindestens eine* $(d-1)$*-dimensionale Hyperfläche gemeinsam haben.*

Geschlossene, rekursive raumfüllende Kurven
Sowohl die Hilbertkurve, als auch alle Kurven vom Peanotyp (auch Peano-Meander) sind demnach geschlossene, rekursive raumfüllende Kurven. Wir werden dies auch für die drei- und höherdimensionalen Variante fordern.

Geschlossene, rekursive raumfüllende Kurven haben stets folgende Eigenschaften:

- Sie sind *gleichmäßig stetig*, d.h. die Geschlossenheit und Rekursivität genügt bereits zum Beweis der (gleichmäßigen) Stetigkeit. Später werden wir zeigen, dass sie sogar der strengeren Hölder-Stetigkeit (mit Exponent $1/d$) genügen.

- Sie sind *nachbarschaftserhaltend* – dies folgt direkt aus der Geschlossenheit. Wir werden ebenfalls noch sehen, dass die Erhaltung von Nachbarschaftsbeziehungen gerade durch die Hölder-Stetigkeit quantifiziert werden kann.

- Sie sind *durch Grammatiken erzeugbar* – damit lässt sich nach bekanntem Schema ein Algorithmus zur Traversierung ableiten.

- Sie sind durch eine geeignete *Arithmetisierung beschreibbar* – dies liefert Algorithmen zur Berechnung der Parameter aus den Punkten und umgekehrt.

6.2 Dreidimensionale Hilbertkurven

Zur Konstruktion einer dreidimensionalen Hilbertkurve wollen wir neben Rekursivität und Geschlossenheit noch folgende typische Konstruktionsmerkmale der zweidimensionalen Hilbertkurve beibehalten:

- Die 3D-Hilbertkurve entsteht durch rekursives Unterteilen des Einheitswürfels in kongruente Teilwürfel mit halber Seitenlänge – also gerade in acht kongruente Teilwürfel;

- Entsprechend wird die zugehörige Intervallschachtelung durch rekursives achteln des Einheitsintervalls erzeugt.

Diese Forderungen liefern uns drei mögliche Grundformen, die in Abbildung 6.1 dargestellt sind.

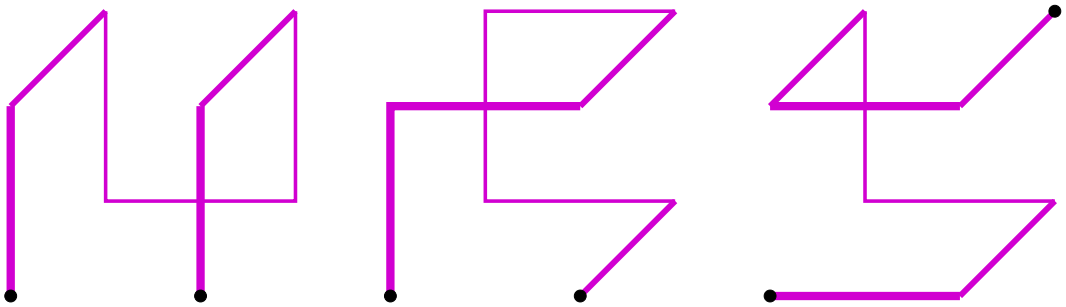


Abbildung 6.1: Mögliche Grundformen zur Konstruktion einer dreidimensionalen Hilbertkurve

Dabei erkennen wir, dass in der rechts abgebildeten „Schraubenform“ Start- und Endpunkt des Leitmotivs nicht an einer gemeinsamen Kante liegen, sondern (raum-)diagonal gegenüber. Dieser diagonale Fall kann im zweidimensionalen Fall nicht auftreten. Daher werden wir diesen zunächst nicht weiterverfolgen, obwohl er prinzipiell zur Konstruktion einer raumfüllenden Kurve geeignet ist. Auch den mittleren Fall klammern wir zunächst aus und konzentrieren uns auf das linke Grundmuster – dasjenige, das dem 2D-Grundmuster am ähnlichsten ist.

6.2.1 Unterschiedliche Konstruktionen einer 3D-Hilbertkurve

Selbst wenn wir uns auf das linke Grundmuster aus Abbildung 6.1 beschränken, ist die Situation lange nicht so übersichtlich wie im zweidimensionalen Fall. Wir erhalten keine eindeutige „Hilbertkurve“, sondern vielmehr eine Vielzahl von raumfüllenden Kurven, die wir alle als dreidimensionale Hilbertkurve bezeichnen können. Im Folgenden wollen wir die möglichen Varianten aufzeigen.

Varianten durch „Kippen“ des Grundmusters

In Abbildung 6.2 ist illustriert, wie durch „Kippen“ eines Grundmusters neue Varianten erstellt werden können. Bei der Konstruktion der Iterationen der Hilbertkurve ist stets ein Teilproblem, zwei aufeinander folgende Eckpunkte des approximierenden Polygons durch ein entsprechendes Grundmuster zu verbinden. Abbildung 6.2 zeigt aber, dass es stets zwei Möglichkeiten gibt, das einzubauende Grundmuster zu orientieren. Abbildung 6.3 zeigt jeweils die zweite Iteration zweier verschiedener Hilbertkur-

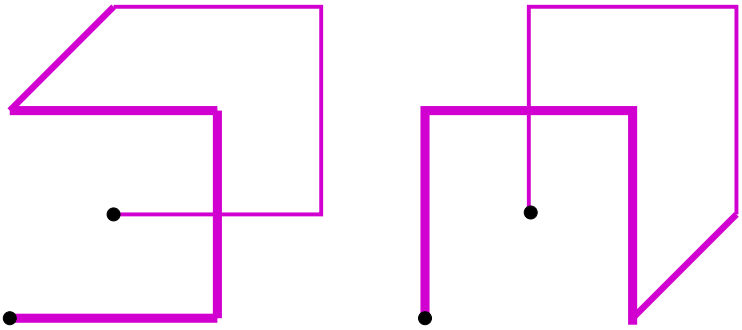


Abbildung 6.2: Variation einer Grundform durch „Kippen“

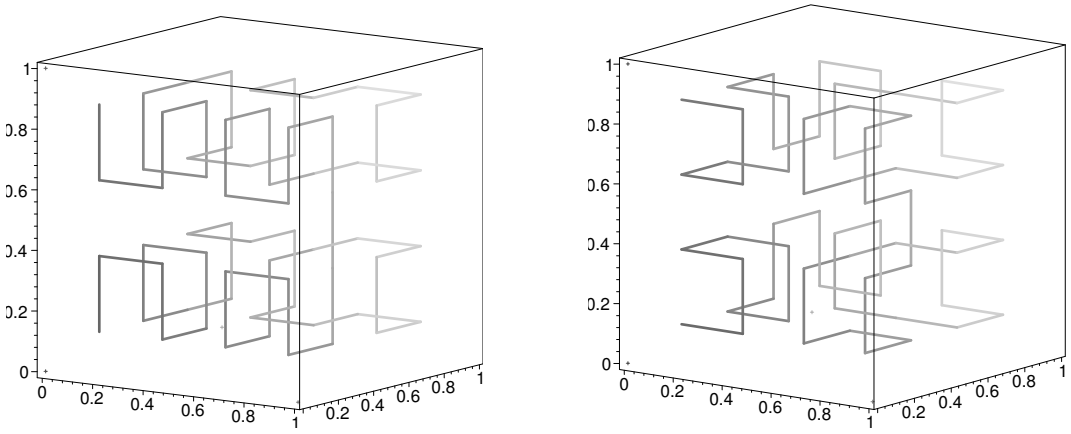


Abbildung 6.3: Zwei verschiedene Konstruktionen einer dreidimensionalen Hilbertkurve (jeweils zweite Iteration)

ven – dabei ist die Orientierung in keinem der acht Grundmuster gleich. Trotzdem werden die acht Teilwürfel in der gleichen Reihenfolge abgelaufen. Die erste Iteration der beiden Kurven ist also noch identisch.

Man überlegt sich leicht, dass es $2^8 = 256$ verschiedene Möglichkeiten gibt, die Orientierungen der Grundmuster auszuwählen.

Variation des approximierenden Polygons

In Abbildung 6.4 ist verdeutlicht, dass sich auch über das Variieren des approximierenden Polygons neue Varianten der Hilbertkurve konstruieren lassen. Es sind jeweils das approximierende Polygon (erste Iteration) und die erste Iteration der zugehörigen Hilbertkurve dargestellt. Konzentrieren

muss man sich dabei auf die Übergänge zwischen dem dritten, vierten und fünften Teilwürfel, sowie zwischen dem fünften, sechsten und siebten Teilwürfen. In beiden Situationen bestehen zwei Möglichkeiten, den Verlauf des approximierenden Polygons zu wählen.

Man beachte, dass die Reihenfolge der Teilwürfel dabei wieder nicht verändert wird. Wir sehen also, dass wir im Dreidimensionalen eine enorme Anzahl verschiedener Hilbertkurven konstruieren können, obwohl wir uns bisher auf nur eines von drei denkbaren Grundmustern beschränkt haben.

6.2.2 Arithmetisierung der 3D-Hilbertabbildung

Die Arithmetisierung der 3D-Hilbertkurve erfolgt nach dem bereits bekannten und erprobten Schema. Da nun eine rekursive Teilung von Intervallen und Würfeln in jeweils acht Teile vorliegt, basiert die Arithmetisierung auf dem Oktalsystem.

Sei der Parameter $t = 0_8.k_1k_2k_3k_4\ldots$ also im Oktalsystem gegeben, dann suchen wir für eine dreidimensionale Hilbertkurve folgende Darstellung:

$$h(0_8.k_1k_2k_3k_4\ldots) = H_{k_1} \circ H_{k_2} \circ H_{k_3} \circ H_{k_4} \circ \cdots \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Die Operatoren H_0 bis H_7 beschreiben dabei jeweils eine Transformation eines der acht Teilwürfel in den Einheitswürfel. Für die im linken Teilbild

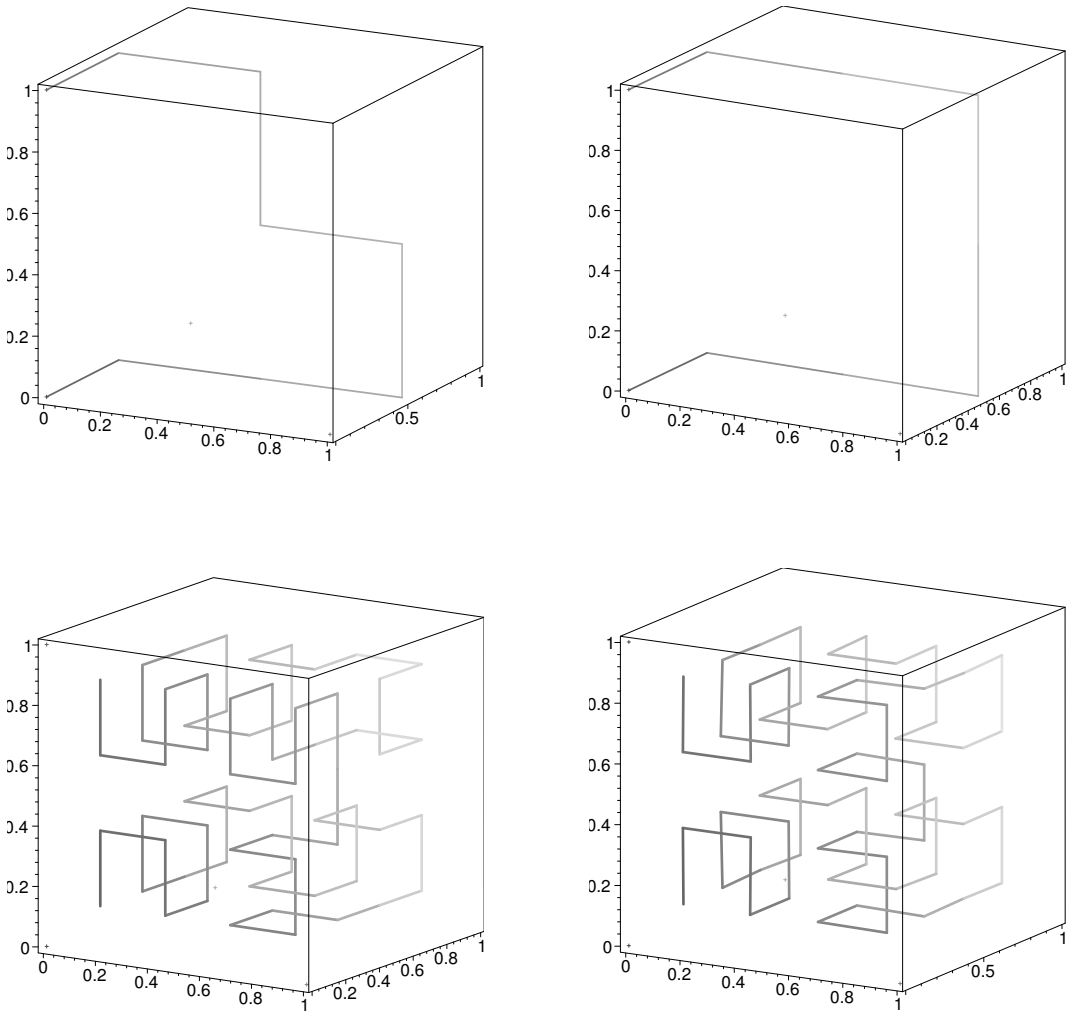


Abbildung 6.4: Die unteren Bilder zeigen zwei dreidimensionale Hilbertkurven (jeweils erste Iteration) mit unterschiedlichen approximierenden Polygonen. Die zugehörigen Polygone (erste Iteration) sind jeweils darüber abgebildet.

der Abbildung 6.3 gezeigte Hilbertkurve erhalten wir folgende Operatoren:

$$H_0\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + 0 \\ \frac{1}{2}z + 0 \\ \frac{1}{2}y + 0 \end{pmatrix} \qquad H_1\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}z + 0 \\ \frac{1}{2}y + \frac{1}{2} \\ \frac{1}{2}x + 0 \end{pmatrix}$$

$$H_2\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \\ \frac{1}{2}z + 0 \end{pmatrix} \qquad H_3\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}z + \frac{1}{2} \\ -\frac{1}{2}x + \frac{1}{2} \\ -\frac{1}{2}y + \frac{1}{2} \end{pmatrix}$$

$$H_4\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{2}z + 1 \\ -\frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \end{pmatrix} \qquad H_5\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \\ \frac{1}{2}z + \frac{1}{2} \end{pmatrix}$$

$$H_6\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{2}z + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \\ -\frac{1}{2}x + 1 \end{pmatrix} \qquad H_7\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + 0 \\ -\frac{1}{2}z + \frac{1}{2} \\ -\frac{1}{2}y + 1 \end{pmatrix}$$

Analog zu den zweidimensionalen Hilbert- und Peano-Arithmetisierungen können wir daraus wieder die beiden Algorithmen für die Berechnung der 3D-Hilbertabbildung und des 3D-Hilbertindex ableiten.

7 Parallelisierung mit Raumfüllenden Kurven

7.1 Parallelisieren im Wissenschaftlichen Rechnen

Die größten und schnellsten gebauten Hochleistungsrechner werden heutzutage vor allem im Bereich des Wissenschaftlichen Rechnens eingesetzt, also bei der Simulation von Phänomenen und Prozessen aus Naturwissenschaft und Technik. Anwendungsbeispiele finden sich in so verschiedenen Bereichen wie der Berechnung der aerodynamischen Eigenschaften von Fahr- und Flugzeugen, der Vorhersage von Wetter- und Klimaphänomenen, oder auch in der molekularen Physik und Chemie.

Die Genauigkeit der erzielten Simulationsergebnisse hängt in der Regel direkt davon ab, wie viele Freiheitsgrade im berechneten Modell verwendet werden können. Zum Beispiel ist in der Aerodynamik eine möglichst feine räumliche Auflösung für genaue Ergebnisse erforderlich. Je nach Umfang der mathematischen Modelle und je nach gewünschter Genauigkeit müssen Simulationsprogramme daher nicht selten mehrere Millionen oder gar Milliarden Variablen bewältigen können. Sowohl die dazu erforderliche Rechenzeit als auch der benötigte Speicherplatz erfordern es, dass die Rechnungen auf Hochleistungsrechnern durchgeführt werden, die heutzutage fast ausschließlich Parallelrechner sind.

Aus Sicht des Programmierers unterscheiden sich Parallelrechner vor allem darin, ob er die Parallelisierung von Daten und Instruktionen selbst erledigen muss, oder ob dies zumindest teilweise durch das Laufzeitsystem geschehen kann. Selbst auf einem verteilten Rechensystem kann durch Hardware oder Software ein virtueller, gemeinsamer Speicher geschaffen werden, der dem Programmierer die Verteilung der Daten auf unterschiedliche Rechenknoten abnimmt. Insbesondere bei massiv parallelen Rechensystemen, also bei Rechnern mit sehr großer Zahl von Rechenknoten mit

eigenem Speicher und Rechenwerk, können aber meist bessere Leistungsdaten erzielt werden, wenn die Parallelisierung explizit durch den Programmierer erfolgt.

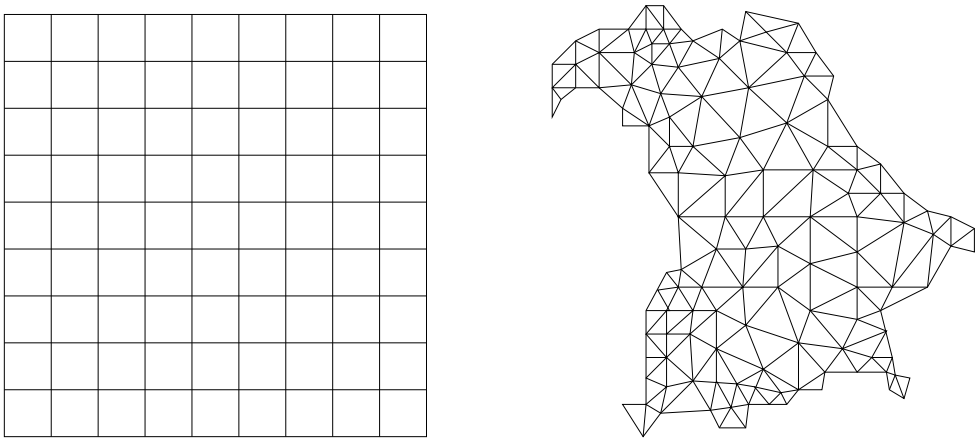
7.1.1 Beispiel: Temperaturverteilung auf einer Metallplatte

Als einfaches Beispiel für eine Problemstellung aus dem wissenschaftlichen Rechnen wollen wir die Berechnung der Temperaturverteilung auf einer Metallplatte untersuchen. Bei diesem Problem sei zunächst die Form der Metallplatte gegeben – dies entspricht mathematisch gesehen unserem Berechnungsgebiet Ω. Desweiteren sei die Temperatur am Rand der Platte gegeben, und für jeden Punkt der Platte seien eventuelle Wärmequellen oder -senken bekannt. Dies soll auch einschließen, ob die Platte von außen Wärme aufnimmt oder abgibt, etwa durch Strahlung oder ähnliches. Die Randbedingungen, sowie die Wärmequellen und -senken werden als zeitlich konstant vorausgesetzt, so dass sich für die Temperatur ein Gleichgewicht einstellen wird. Diese *stationäre* Temperaturverteilung sei gesucht.

Auf dem Computer können wir die Lösung dieses Problems angehen, indem wir die Platte mit einem Gitter von Messpunkten überziehen. An diesen Messpunkten soll die Temperatur berechnet werden. In Abbildung 7.1 sind zwei Beispiele für solche Rechengitter abgebildet. Das linke Beispiel zeigt ein regelmäßiges Rechtecksgitter (ein sog. *kartesisches* Gitter), das eine quadratische Metallplatte überzieht. Rechts daneben ist ein Dreiecksgitter für ein komplizierter geformtes Rechengebiet dargestellt. Die Unbekannten liegen jeweils auf den Gitterknoten.

Die Näherung, die Temperatur nur noch an den Gitterpunkten zu berechnen, reduziert das kontinuierliche Problem, eine Temperaturfunktion zu finden, auf ein Problem mit endlich vielen Unbekannten. Zur Berechnung der Temperaturen an den Messpunkten stellen wir ein Gleichungssystem für diese Unbekannten auf. Dabei wissen wir aus der Physik, dass sich in Abwesenheit von Temperaturquellen und -senken die Temperatur an einem Gitterpunkt auf das Mittel der umliegenden Temperaturen einstellen wird. Bei den beiden Gittern aus Abbildung 7.1 können wir zur Mittelung jeweils die Punkte heranziehen, die mit dem aktuellen Gitterpunkt durch Gitterlinien direkt verbunden sind.

In dem kartesischen Gitter können wir ferner in guter Näherung voraussetzen, dass die Temperatur als das arithmetischen Mittel der Tempe-



(a) kartesisches Gitter

(b) Dreiecksgitter

Abbildung 7.1: Zwei Beispiele für Rechengitter: links ein regelmäßiges, *kartesisches* Gitter, rechts ein Dreiecksgitter auf einem komplizierten Berechnungsgebiet.

raturen an den vier direkt benachbarten Gitterpunkten bestimmt werden kann. Bezeichnen wir die Unbekannte in der i-ten Zeile und j-ten Spalte des kartesischen Gitters mit $u_{i,j}$, dann erhalten wir die Gleichungen

$$u_{i,j} = \frac{1}{4}\left(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}\right)$$

für alle i und j. Die Gleichungen können wir auch schreiben als

$$u_{i,j} - \frac{1}{4}\left(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}\right) = 0.$$

Eventuelle Wärmequellen oder -senken können wir dann durch eine zusätzliche rechte Seite $f_{i,j}$ einbringen:

$$u_{i,j} - \frac{1}{4}\left(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}\right) = f_{i,j} \qquad \text{für alle } i, j. \tag{7.1}$$

Durch Lösen dieses Gleichungssystems erhalten wir eine Näherung für die exakte Lösung, die umso genauer sein wird, je mehr Gitterpunkte wir verwenden.

7.1.2 Parallelisierung der Lösung

Da wir unsere Temperaturverteilung natürlich möglichst exakt berechnen wollen, versuchen wir die Zahl der Unbekannten so groß wie möglich zu machen. Um das Problem auf einem parallelen (Groß-)Rechner bearbeiten zu können, zerlegen wir unsere Unbekannten in kleinere Teile, die sogenannten *Partitionen*. Jede Partition soll auf einem eigenen Rechenknoten (Prozessor samt eigener Hauptspeicher) bearbeitet werden. Natürlich werden nicht alle Partitionierungen zu gleich schnellem Code führen. Welches aber sind die Kriterien für eine effiziente Partitionierung?

Anforderungen an die Partitionierung
Typische „Effizienz-Killer“ beim parallelen Rechnen sind zum Einen der im Vergleich zur Prozessorgeschwindigkeit viel langsamere Datenaustausch zwischen den Prozessoren (Kommunikation), sowie zum Anderen eine ungleichmäßige Auslastung aller Prozessoren (Lastverteilung). Für die Gebietszerlegung in unserem Anwendungsfall ergeben sich dadurch die folgenden, fundamental wichtigen Anforderungen:

- *Jede Partition soll annähernd gleich viele Unbekannte enthalten!*

  Da der Rechenaufwand in den meisten Fällen hauptsächlich von der Zahl der Unbekannten abhängt, ist damit sicher gestellt, dass jeder Prozessor in etwa gleich viel Arbeit zugeteilt bekommt (Lastverteilung, *load balancing*). Alle Prozessoren sind dann gleichmäßig und optimal ausgelastet. Keine Rechenzeit geht dadurch verloren, dass ein Teil der Prozessoren bereits früher fertig ist als die anderen und dann ungenutzt bleibt.

- *Die Partitionsgrenzen sollen möglichst kurz sein!*

  In unserem Gleichungssystem sind nur direkt benachbarte Unbekannte miteinander gekoppelt. Für alle typischen Berechnungen (Gleichungssysteme lösen, Fehler ermitteln) brauchen wir deshalb unmittelbar benachbarte Unbekannte stets gleichzeitig. Deshalb sollen für möglichst viele Unbekannte alle Nachbarn in der gleichen Partition liegen. Dies *vermeidet Kommunikation* zwischen den einzelnen Prozessen, um die aktuellen Werte von Unbekannten auszutauschen bzw. konsistent zu halten.

Im geometrischen Sinn sollen daher möglichst wenige Unbekannte an Partitionsgrenzen liegen. Dies erreichen wir, wenn die Grenzen selbst möglichst kurz und die Partitionen entsprechen kompakt sind.

Darüber hinaus können je nach Anwendungsfall noch weitere Forderungen hinzukommen. Bei *adaptiven* Gittern, zum Beispiel, wird an bestimmten kritischen Stellen die Zahl der Gitterpunkte erhöht. Dadurch kann etwa an der kritischen Stelle die Genauigkeit des Ergebnisses verbessert werden. Die Verfeinerung kann aber auch aus numerischen Gründen erforderlich werden, damit zum Beispiel große Fehler an komplizierten Randstrukturen nicht die Lösung im gesamten Rechengebiet verfälschen. In beiden Fällen kann das Hinzunehmen oder Entfernen von Unbekannten auch während der Berechnung selbst erfolgen. Zum Beispiel könnte ein Algorithmus auf Basis eines *Fehlerschätzers* entscheiden, dass an bestimmten Stellen die Zahl der Unbekannten deutlich zu erhöhen ist.

Adaptive Gitter stellen daher folgende, zusätzliche Anforderungen an das Partitionierungsverfahren:

- Die Partitionierung darf nicht auf rein geometrischer Basis erfolgen, sondern muss die Positionen der Unbekannten direkt berücksichtigen.

- Wird die Lastverteilung durch adaptives Hinzufügen oder Entfernen von Unbekannten unausgewogen, so muss eine *schnelle Re-Partitionierung* möglich sein. Diese darf auf keinen Fall mehr Zeit kosten als das Weiterrechnen mit einer schlechten Lastverteilung.

- Wenn nur wenige Unbekannte hinzukommen oder wegfallen, soll sich die Form der Partitionen nur wenig ändern. Möglichst viele Unbekannte sollen bei der Re-Partitionierung in ihrer alten Partition verbleiben, da auch das Migrieren von Unbekannten in andere Partitionen teure Kommunikation zwischen den Prozessoren erfordert.

7.2 Parallelisierung mit raumfüllenden Kurven

Raumfüllende Kurven bieten zwei Eigenschaften, die sie für den Einsatz in der Parallelisierung prädestinieren:

- Raumfüllende Kurven schaffen eine *lineare Ordnung* in einem mehrdimensionalen Gebiet. Eine linear aufgezählte Liste in gleich große Stücke zu zerteilen, ist eine triviale Aufgabe, die aber gerade das Partitionierungsproblem löst.

- Raumfüllende Kurven sind *nachbarschaftserhaltend*. Zwei Punkte im Raum, deren Hilbert- oder Peano-Index sich nur wenig unterscheidet, sind auch im Bildraum eng benachbart. Partitionen, die im Indexraum benachbarte Punkte zusammenfassen, werden also auch im Bildraum eine lokale Punktmenge bilden.

Dies lässt erwarten, dass eine Parallisierung mit Hilfe von raumfüllenden Kurven unseren beiden wichtigsten Anforderungen nach guter Lastverteilung und kompakten Partitionen gerecht werden könnte. Die konkrete Durchführung der Partitionierung kann je nach Situation über den Index der Gitterpunkte erfolgen, oder mit Hilfe einer Traversierung.

Partitionierung nach dem Index der Gitterpunkte
Bei der Partitionierung nach Index wird für jeden Gitterpunkt der zugehörige Index der raumfüllenden Kurve berechnet. Die Partitionierung geschieht dann abhängig von den Indizes. Zum Beispiel könnte man die Gitterpunkte nach Index sortieren, und die sortierte Liste (bzw. das sortierte Feld) der Gitterpunkte in gleich große Partitionen zerlegen. Eine prototypische Implementierung, zum Beispiel auf Basis der Peanokurve, wäre dann durch die folgenden Schritte gegeben:

1. Berechne für jeden Gitterpunkt g_i den Peano-Index $\bar{p}^{-1}(g_i)$.

2. Sortiere den Vektor (das Array/die Liste) g der Gitterpunkte nach den Indizes $\bar{p}^{-1}(g_i)$.

3. Zerlege den Vektor g der N Gitterpunkte in K Partitionen:

$$g^{(k)} := \left(g_{(k-1)N/K}, \ldots, g_{kN/K-1}\right) \qquad \text{für } k = 1, \ldots, K.$$

Während Schritt 3 mit linearem Rechenaufwand zu bewältigen ist, verursacht das Sortieren mindestens einen Rechenaufwand von der Ordnung $\mathcal{O}(N \log N)$. Die Berechnung der Peano-Indizes scheint auf den ersten Blick linearen Aufwand zu besitzen. Allerdings dürfen wir nicht vergessen, dass

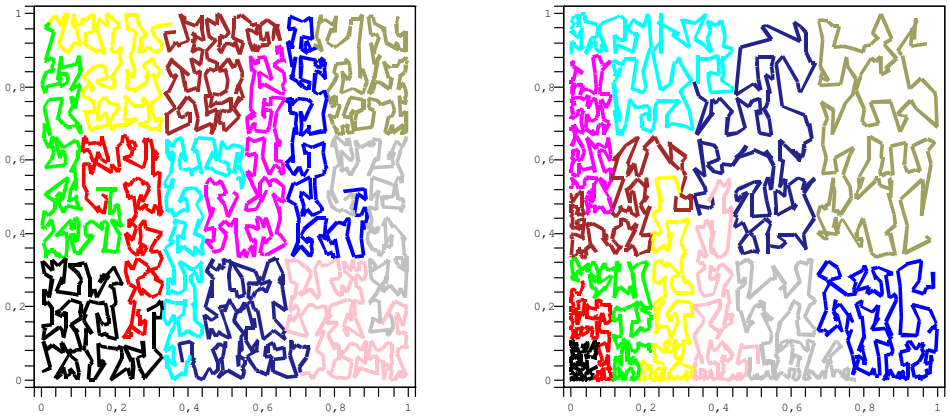


Abbildung 7.2: Partitionierung einer Menge zufällig gewählter Gitterpunkte durch Sortieren nach Peano-Index. Im rechten Beispiel sind die Punkte stark in Richtung linke untere Ecke verdichtet.

die Berechnung von $\bar{p}^{-1}(g_i)$ von der Anzahl der berücksichtigten Stellen in g_i abhängt. Bei steigender Zahl N von Gitterpunkten muss je nach Verteilung der Gitterpunkte auch die Anzahl der berücksichtigten Stellen wachsen. Im schlechtesten Fall sollten $\mathcal{O}(\log N)$ ausreichend sein. Damit erhalten wir insgesamt einen Rechenaufwand von $\mathcal{O}(N \log N)$ für das Partitionierungsproblem. Als Vorteil des Verfahrens dürfen wir verbuchen, dass wir keinerlei Einschränkungen bezüglich der Struktur des Gitters machen müssen. Selbst völlig strukturlose Punktmengen wie in Abbildung 7.2 können wir problemlos in Partitionen zerlegen.

Partitionierung mit Hilfe einer Traversierung

Für ein regelmäßiges Gitter, wie das Rechtecksgitter in Abbildung 7.1, muss eine Partitionierung mittels Sortieren fast zwangsläufig ineffizient sein, da die spezielle Struktur des Gitters nicht ausgenutzt wird. Stattdessen können wir zum Beispiel von einer Traversierung ausgehen, wie sie in Kapitel 3.2 besprochen wird. Ein prototypischer Algorithmus zur Partitionierung besteht dann zum Beispiel aus folgenden Schritten:

1. Traversiere die Gitterpunkte g_i entlang einer Peanokurve und nummeriere die Gitterpunkte dabei fortlaufend.

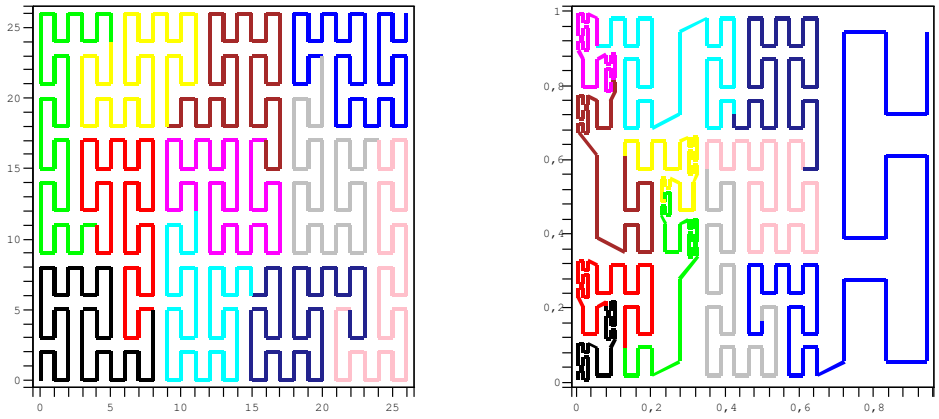


Abbildung 7.3: Partitionierung zweier Rechtecksgitter mit Hilfe einer Traversierung entlang der Peanokurve. Rechts ein Beispiel mit adaptiver Verfeinerung.

2. Traversiere die N Gitterpunkte nochmals und ordne jeden Gitterpunkt in Abhängigkeit von seiner fortlaufenden Nummer n_i der Partition k zu, so dass
$$\frac{(k-1)N}{K} \leq n_i < \frac{kN}{K}.$$

Der Partitionierungsalgorithmus hat den Rechenaufwand $\mathcal{O}(N)$, da bei der Traversierung jeder Gitterpunkt genau einmal bearbeitet wird. Sowohl das Nummerieren eines Gitterpunkts als auch das Zuordnen zu einer Partition kann mit konstantem Rechenaufwand implementiert werden. Damit ist die Partitionierung durch Traversieren günstiger als die mittels Sortieren. Wir erkaufen uns diesen Vorteil dadurch, dass das Gitter eine bekannte Struktur aufweisen muss. Es muss aber nicht notwendig ein regelmäßiges Rechtecksgitter sein, wie Abbildung 7.3 zeigt.

7.3 Hölder-Stetigkeit raumfüllender Kurven

Die allgemeinen Eigenschaften raumfüllender Kurven und die Resultate in der Praxis (siehe etwa Abbildung 7.2 und 7.3) legen nahe, dass raumfüllende Kurven eine gute Heuristik für die Aufteilung von Daten in kompakte

Partitionen liefern. Lässt sich diese Eigenschaft auch durch eine mathematische Eigenschaft der raumfüllenden Kurven charakterisieren oder gar quantifizieren?

Wir werden in diesem Abschnitt untersuchen, dass der Begriff der *Hölder-Stetigkeit* genau dafür geeignet ist.

Definition 7.1 (Hölder-stetig)
Eine Abbildung $f\colon I \to \mathbb{R}^n$ heißt auf dem Intervall I Hölder-stetig *zum Exponent r, wenn es ein $C > 0$ gibt, so dass für alle $x, y \in I$ gilt:*

$$\|f(x) - f(y)\|_2 \leq C\,|x - y|^r$$

Anstelle der euklidischen Norm $\|\cdot\|_2$ können wir natürlich auch jede äquivalente Norm zur Definition heranziehen.

Im Hinblick auf die raumfüllenden Kurven schafft die Hölder-Stetigkeit eine Beziehung zwischen dem Abstand der Indizes und dem Abstand der Bildpunkte. Der Abstand der Indizes entspricht dem Parameterabstand $|x - y|$. Der Abstand der Bildpunkte ist gerade $\|f(x) - f(y)\|_2$. Wir werden zeigen, dass die Hilbertkurve gerade Hölder-stetig zum Exponent $r = d^{-1}$ ist, wobei d die Dimension bezeichnet:

$$\|f(x) - f(y)\|_2 \leq C\,|x - y|^{1/d} = C\sqrt[d]{|x - y|}$$

7.3.1 Hölder-Stetigkeit der 3D-Hilbertkurve

Der Beweis der Hölder-Stetigkeit der 3D-Hilbertkurve ist bis auf einige technische Details identisch zum Beweis der gleichmäßigen Stetigkeit der 2D-Kurve:

1. Seien zwei beliebige Parameter $x, y \in \mathcal{I}$ gegeben. Dann lässt sich stets ein n finden, so dass $8^{-(n+1)} \geq |x - y| < 8^{-n}$.

2. 8^{-n} ist gerade die Intervalllänge der Intervallschachtelung in der n-ten Iteration. Daraus folgt, dass das Intervall $[x, y]$ maximal zwei mögliche Intervalle der n-ten Schachtelungstiefe überlappt. Diese beiden Intervalle müssen zudem benachbart sein!

3. Aufgrund der Konstruktion der 3D-Hilbertkurve werden diese beiden Intervalle auf zwei benachbarte Teilwürfel der Seitenlänge 2^{-n} abgebildet. Die Funktionswerte $h(x)$ und $h(y)$ liegen folglich in dem Quader, der durch diese beiden Würfeln gebildet wird.

4. Der maximale Abstand von $h(x)$ und $h(y)$ kann daher nicht größer sein als die Länge der Raumdiagonale dieses Quaders. Diese beträgt gerade
$$2^{-n} \cdot \sqrt{1^2 + 1^2 + 2^2} = 2^{-n} \cdot \sqrt{6}$$

5. Für den Abstand $\|f(x) - f(y)\|_2$ gilt daher folgende Ungleichung:
$$\begin{aligned} \|h(x) - h(y)\|_2 &\leq 2^{-n}\sqrt{6} = \left(8^{-n}\right)^{1/3} \sqrt{6} = \left(8^{-(n+1)}\right)^{1/3} 8^{1/3}\sqrt{6} \\ &\leq 2\sqrt{6}\,|x-y|^{1/3} \end{aligned}$$
Mit $C := 2\sqrt{6}$ und $d = 3$ ist damit die Hölder-Stetigkeit bewiesen.

An der Beweisführung erkennen wir, dass wiederum nur die Eigenschaften der Geschlossenheit und der Rekursivität für den Beweis erforderlich sind. Alle geschlossenen, rekursiven raumfüllenden Kurven sind demnach Hölder-stetig zum Exponent d^{-1}.

7.3.2 Hölder-Stetigkeit und Parallelisierung

Die Hölder-Stetigkeit schafft über die Ungleichung
$$\|f(x) - f(y)\|_2 \leq C \sqrt[d]{|x-y|}$$
eine Beziehung zwischen dem Abstand $|x-y|$ der durch die raumfüllende Kurve definierten Indizes und dem Abstand $\|f(x) - f(y)\|$ der zugehörigen Punkte. Der Abstand $|x-y|$ entspricht zunächst einer „abgelaufenen Strecke“ auf der Kurve. Fassen wir jedoch die Punkte als Teile eines räumlichen Punktgitters oder auch als Repräsentanten eines Zellgitters auf, dann ist der abgelaufenen Teilkurve ein gewisses Volumen zugeordnet. Die Hölder-Stetigkeit liefert also ein Verhältnis zwischen Volumen und Ausdehnung (z.B. Durchmesser) einer Partition. In diesem Sinne liefert die Hölder-Stetigkeit also eine Art Maß für die *Kompaktheit* der Partitionen.

8 Spacetrees und Raumfüllende Kurven

8.1 Quartalbäume, Oktalbäume und Spacetrees als Mittel zur Geometriebeschreibung

Techniken zur effizienten Beschreibung und Modellierung der Geometrie von starren Objekten werden in der Informatik meist im Rahmen der Grafischen Datenverarbeitung besprochen. Wichtige Anwendungsgebiete erschließen sich dort zum Beispiel im *computer aided design* (CAD), also dem computerunterstützten Entwurf von Bauteilen. Im Fahrzeugbau könnten dies etwa Motor- und Karosserieteile sein, oder auch ein komplettes Fahrzeug.

Eingang in das wissenschaftliche Rechnen finden diese Techniken immer dann, wenn es nicht nur um die äußerliche Darstellung von Bauteilen geht. Soll zum Beispiel die mechanische Belastbarkeit eines Bauteils oder die Aerodynamik eines Fahrzeugs im Computer modelliert und berechnet werden, dann ist ebenfalls eine exakte Beschreibung der geometrischen Gestalt des Bauteils notwendig.

Generell unterscheidet man in der Geometriemodellierung zwischen *oberflächenbasierten* und *volumenbasierten* Beschreibungsmodellen. Oberflächenbasierte Modelle beschreiben zunächst die Topologie des darzustellenden Objekts, also die Position seiner (Eck-)Punkte, Kanten und Flächen, sowie deren Nachbarschaftsbeziehungen. Einfachstes Modell dieser Art ist das sogenannte *Drahtgittermodell*, das mittlerweile jedoch nur noch in alten Science-Fiction-Filmen anzutreffen sein dürfte. Weiterentwickelte Modelle, die es erlauben die Kanten und Flächen mit sogenannten Freiformkurven oder -flächen zu beschreiben, sind jedoch auch heute noch Stand der Technik im CAD-Bereich.

Im Gegensatz dazu beschreiben volumenbasierte Modelle das Objekt

nicht durch seine Oberfläche, sondern durch den von ihm eingenommenen Raum. Das einfachste solche Verfahren ist das *Normzellenschema*. Hier wird das zu beschreibende Objekt in ein regelmäßiges Gitter aus würfel- oder quaderförmigen Zellen eingebettet. Für jede Zelle wird dann gespeichert, ob sie sich innerhalb oder außerhalb des Körpers befindet. Diese zellorientierte Beschreibung ergänzt sich hervorragend mit vielen numerischen Verfahren im Bereich des wissenschaftlichen Rechnens. Auch hier wird im Rahmen einer *Diskretisierung* die zu berechnende Größe nicht als kontinuierliche Funktion berechnet. Stattdessen verwendet man ein Rechengitter, und berechnet die Größe an diskreten Gitterpunkten, die je nach Verfahren innerhalb der Gitterzellen oder auch auf den Kanten oder Eckpunkten der Zellen liegen können.

Während die typischen oberflächenorientierten Beschreibungen umso umfangreicher werden, je komplizierter das Objekt aufgebaut ist, hängt die Genauigkeit der Beschreibung beim Normzellenschema ausschließlich von der Größe der Gitterzellen ab. Die gewählte Auflösung bestimmt somit die zur Beschreibung eines Objekts notwendige Speichermenge – für jede Zelle ist zumindest ein Bit erforderlich. Müssen unterschiedliche Materialeigenschaften gespeichert werden, steigt der Speicherverbrauch pro Normzelle natürlich an.

Ein Rechenbeispiel soll uns helfen, die Größe des erforderlichen Speicherbedarfs festzustellen. Zu modellieren sei etwa die Karosserie eines Autos. Wir gehen von einer Länge von ungefähr 4 m aus, bei einer Breite und Höhe von jeweils ca. 1,5 m. Bei einer gleichmäßigen Auflösung von 1 cm benötigen wir bereits $400 \times 150 \times 150$ Gitterzellen – also etwa 9 Millionen. Obwohl die Datenmenge damit bereits im Megabyte-Bereich angesiedelt ist, reicht die Darstellung des Autos als Konglomerat von 1 cm großen Klötzchen für die meisten Anwendungen noch bei weitem nicht aus. Zum Überprüfen, ob sich zum Beispiel Türen und Motorhaube des Autos öffnen und schließen lassen, ohne dass es zu Kollisionen kommt, würde sich kein Ingenieur mit einer Toleranz von 1 cm zufrieden geben. Noch ließe sich die Aerodynamik des Autos ausreichend genau berechnen. Erhöhen wir die Auflösung jedoch auf 1 mm, so erhöht sich die Zahl der Gitterzellen um den Faktor $10 \times 10 \times 10 = 10^3$ auf nunmehr 9 Milliarden Zellen. Für eine normale Workstation wird diese Datenmenge bereits unhandlich.

8.1.1 Quartalbäume und Oktalbäume

Die sogenannten *Quartalbäume* (im Zweidimensionalen) und *Oktalbäume* (im Dreidimensionalen) sind Beschreibungsmodelle, die das rasante Anwachsen der Anzahl der Zellen beim Normzellenschema begrenzen sollen. Den Modellen liegt die Idee zugrunde, die Auflösung der Gitterzellen nur dort zu erhöhen, wo dies auch erforderlich ist. Dazu wird ein rekursiver Ansatz verfolgt:

1. Ausgangspunkt ist ein Gitter mit sehr grober Auflösung – in der Regel beginnt man mit einem Gitter, das nur aus einer einzigen Zelle besteht. In diese wird das gesamte zu beschreibende Objekt eingebettet.

2. In einem rekursiven Prozess werden nun die Gitterzellen schrittweise solange verfeinert, bis entweder alle Gitterzellen ganz innerhalb oder außerhalb des Objekts liegen, oder eine vorgegebene minimale Zellgröße (die maximal gewünschte Auflösung) erreicht haben.

3. Dabei werden alle Zellen, die ganz innerhalb oder außerhalb des Objekts liegen, nicht mehr weiter verfeinert.

4. Alle anderen Zellen – die somit gerade einen Abschnitt des Objektrands enthalten – werden weiter in kleinere Gitterzellen zerlegt, sofern die maximale Auflösung noch nicht erreicht ist.

Normalerweise werden die Gitterzellen dabei in kongruente Teilzellen zerlegt. Zum Beispiel bietet sich bei einem zweidimensionalen, quadratischen Gitter die Zerlegung der quadratischen Zellen in jeweils vier Teilquadrate mit halber Seitenlänge an – Abbildung 8.1 illustriert das schrittweise Erzeugen eines solchen rekursiven Gitters an einem einfachen Beispiel.

Als Datenstruktur zur Beschreibung des Gitters bietet sich eine Baumstruktur an: jede Gitterzelle entspricht einem Knoten des Baumes. Die Ausgangszelle bildet die Wurzel. Die Teilzellen jeder Gitterzelle sind entsprechend die Tochterknoten im Baum. Für ein zweidimensionales, rekursiv erzeugtes Gitter ergibt sich somit ein Baum, in dem jeder Knoten vier Töchter hat. In Abbildung 8.1 ist diese Baumstruktur für das Beispielgitter angegeben. Entsprechend der vierfachen Verzweigungsstruktur werden derartige Gitter und ihre zugehörigen Bäume als *Quadtrees* (Quartalbaum) bezeichnet.

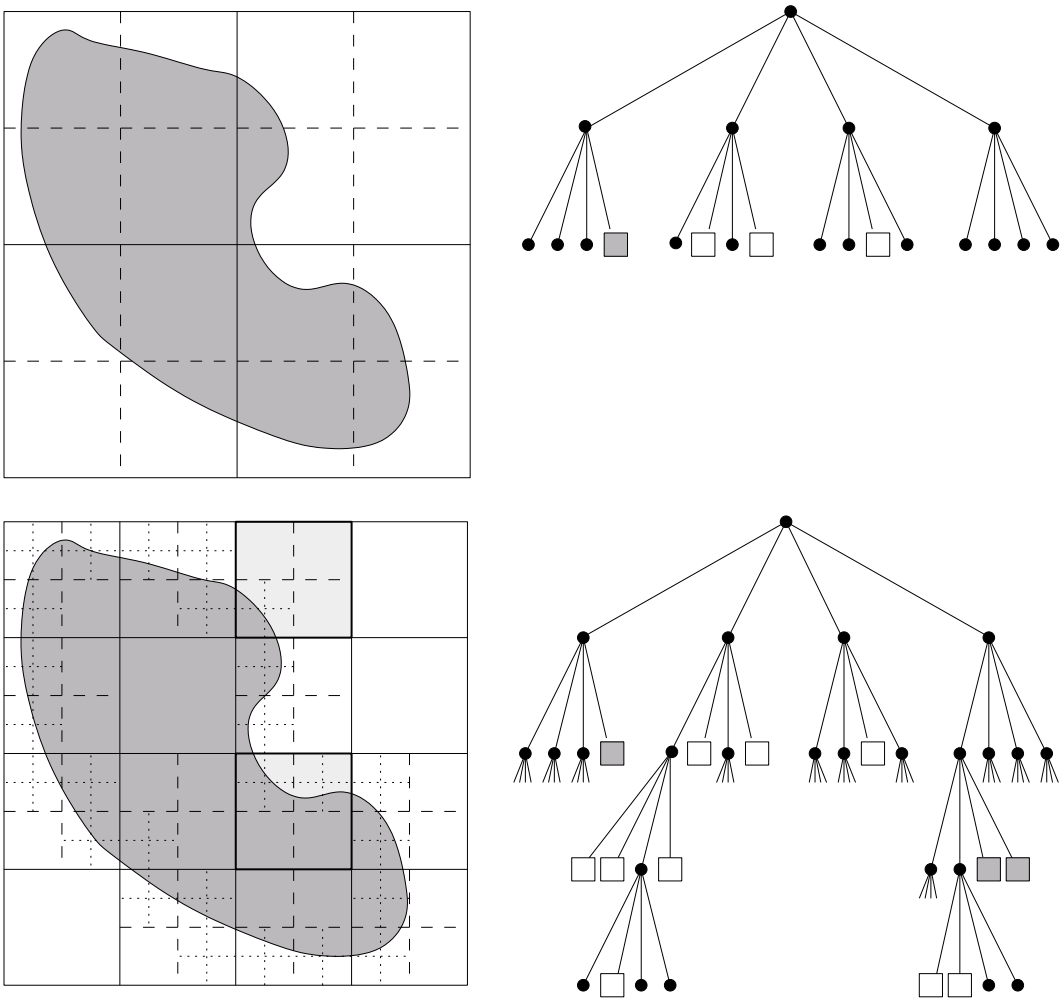


Abbildung 8.1: Schrittweises Erzeugen der Quartalbaumdarstellung eines zweidimensionalen Gebiets. Der Quartalbaum ist nur für die beiden markierten Zellen vollständig angegeben.

Die Gittererzeugung lässt sich ganz analog auf der dreidimensionalen Fall erweitern. Die Gittererzeugung basiert dann auf der rekursiven Zerlegung von Würfelzellen in acht Teilwürfel. Wir erhalten entsprechend einen achtfach verzweigten Baum und sprechen daher von *Octrees* bzw. von *Oktalbäumen*.

8.1.2 Spacetrees

Anstatt die Seitenlänge der Gitterzelle von Baumebene zu Bauebene zu halbieren, können auch rekursiv verfeinerte Gitter erzeugt werden, die die Zellen dritteln oder vierteln (und so weiter). Ein zweidimensionales „Drittelungsgitter“ würde entsprechend durch einen Baum mit jeweils neun Tochterknoten beschrieben – einem „Nonalbaum“. Spätestens im dreidimensionalen Fall, wo jeweils 27 Tochterknoten auftreten, müssen wir uns jedoch nach einer handlicheren Bezeichnung als „Vicenoseptenärbaum“ umsehen:

Definition 8.1 (k^d-Spacetree)
Ein k^d-Spacetree ist ein (evtl. attributierter[1]) Baum mit folgenden Eigenschaften:

- *Jeder Knoten des Baumes ist entweder ein Blatt oder besitzt genau k^d Unterbäume. Jeder Unterbaum ist wieder ein k^d-Spacetree.*
- *Jeder Knoten repräsentiert einen d-dimensionalen Hyperwürfel. Insbesondere gilt dies also auch für jedes Blatt.*
- *Von Level zu Level des Spacetrees reduziert sich die Seitenlänge der Hyperwürfel um den Faktor k^{-1}.*

Ein Spacetree heiße regelmäßig verfeinert, *wenn alle seine Blätter auf dem gleichen Level liegen. Ansonsten heißt er* adaptiver *Spacetree.*

Ein Quartalbaum ist demnach ein 2^2-Spacetree, ein Oktalbaum ein 3^2-Spacetree. Wir werden den Begriff des Spacetrees hier tatsächlich auf den Fall würfelförmiger Zellen beschränken. Selbstverständlich kann das Prinzip aber auch auf die Unterteilung in Rechteckeoder Quader übertragen werden, wobei auch die Untergebiete nicht mehr geometrisch ähnlich sein müssen.

[1] Attributiert ist ein Baum zum Beispiel dann, wenn in jedem Knoten oder Blatt zusätzliche Informationen gespeichert sind – etwa über Materialeigenschaften, oder ähnliches.

8.1.3 Anzahl der benötigten Gitterzellen im Normzellenschema und in Spacetrees

Rein vom Gefühl her ist klar, dass eine Spacetree-Struktur weniger Zellen zur Beschreibung eines vorgegebenen Objekts benötigt als das Normzellenschema. Schließlich werden nur die Gitterzellen verfeinert, bei denen dies auch wirklich erforderlich ist. Aber lässt sich dieser Unterschied auch quantitativ charakterisieren?

Betrachten wir dazu ein recht einfaches, zweidimensionales Beispiel: als Objekt sei ein achsenparalleles Rechteck gegeben. Wir nehmen an, dass wir das Rechteck in das Einheitsquadrat $[0,1]^2$ einbetten können, ohne dabei übermäßig verschwenderisch mit Gitterzellen umzugehen – das Rechteck sei also nicht wesentlich kleiner als das Einheitsquadrat (vgl. Abbildung 8.2). Wir untersuchen nun, wie stark die Zahl der Zellen steigt, wenn wir die Auflösung schrittweise verdoppeln:

Beim Normzellenschema werden alle Zellen des Gitters durch vier Zellen mit halber Seitenlänge ersetzt. Unabhängig von der Größe des gegebenen Rechtecks erhalten wir somit n^2 Zellen, wenn wir n Zellen je Raumrichtung verwenden. Die Zellen haben dann eine Größe von $h = n^{-1}$. Die Anzahl der Zellen steigt also wie $\mathcal{O}(h^{-2})$ mit der Auflösung h.

In einem Quadtree werden dagegen nur diejenigen Zellen unterteilt, die auf dem Rand des Rechtecks liegen. Wie ändert sich dadurch die Anzahl der Zellen wenn wir die maximale Auflösung halbieren? Ein Quadtree mit maximaler Auflösung $2h$ habe r Zellen, die auf dem Rand liegen und s Zellen, die entweder ganz innerhalb oder ganz außerhalb des Rechtecks liegen. Die r Randzellen werden nun durch 4 Zellen halber Seitenlänge ersetzt. Von diesen $4r$ Zellen liegen aber nur $2r$ Zellen wieder auf dem Rand: entlang der Kanten leuchtet dies unmittelbar ein (vgl. Abbildung 8.2); auf den vier Ecken werden aus den ehemals 4 Zellen nun 16 Gitterzellen, von denen insgesamt aber auch wieder genau 8 auf dem Rand liegen müssen. Der Quadtree mit Auflösung h hat somit $2r$ Randzellen plus $s + 2r$ Zellen, die nicht mehr weiter verfeinert werden.

Für die Anzahl der Gitterzellen s_k und r_k eines k-fach verfeinerten Quadtree-

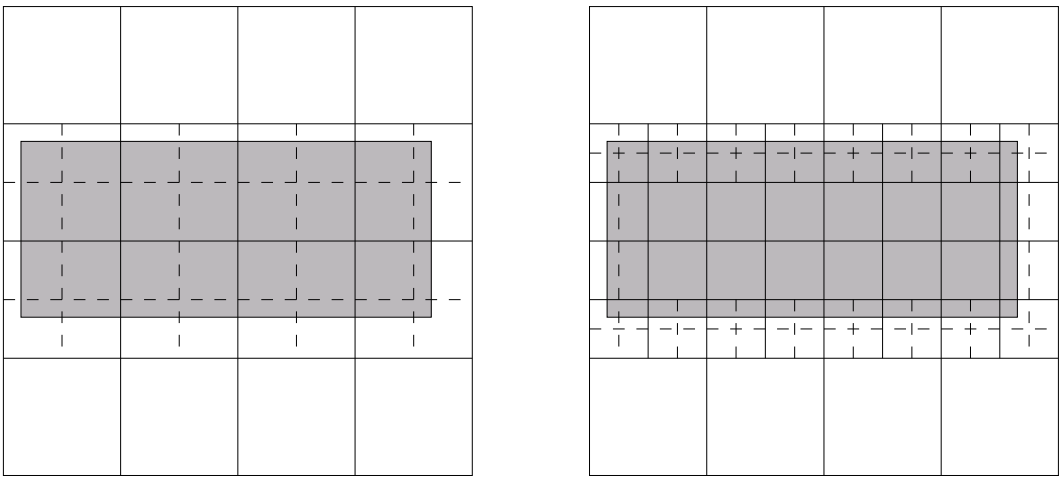


Abbildung 8.2: Verfeinerung eines Quadtree für ein achsenparalleles Rechteck

Gitters erhalten wir somit folgende Rekursionsformel:

$$r_k = 2 \cdot r_{k-1}, \qquad s_k = s_{k-1} + r_{k-1}.$$

Der Fall $k = 0$ entspreche gerade dem Einheitsquadrat, und das Rechteck sei so platziert, dass für $k = 1$, also die erste Unterteilungsstufe, alle vier Zellen Randzellen sind: $s_1 = 0$ und $r_1 = 4$. Damit erhalten wir gemäß der Rekursionsformel für die Zahl der Randzellen

$$r_k = 2 \cdot r_{k-1} = \ldots = \underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{(k-1)\text{-mal}} \cdot r_1 = 2^{k+1},$$

und entsprechend für die Zahl der vollständig inneren oder äußeren Zellen

$$s_k = s_{k-1} + r_{k-1} = s_{k-1} + 2^k = s_{k-2} + 2^{k-1} + 2^k = \ldots = 0 + \sum_{\kappa=1}^{k} 2^\kappa = 2^{k+1} - 1.$$

Ein Quadtree der Tiefe k hat demnach insgesamt $2 \cdot 2^k - 1$ Zellen Diese Zahl bringen wir wieder in Relation zur Gitterweite h: vom Einheitsquadrat ausgehend hat eine k-fach unterteilte Gitterzelle gerade die Seitenlänge $h = 2^{-k}$. Das bedeutet, dass der Quadtree knapp $2 \cdot h^{-1}$ Zellen enthält. Statt mit $\mathcal{O}(h^{-2})$ beim Normzellenschema wächst die Zahl der Zellen also nur noch mit $\mathcal{O}(h^{-1})$.

Ganz allgemein wächst die Anzahl der benötigten Zellen nicht etwa mit dem Volumen des modellierten Körpers, sondern nur mit der Länge seines Randes! Im Dreidimensionalen gilt entsprechend, dass die Zahl der Zellen von der Größe der Oberfläche abhängt. In beiden Fällen gewinnt man eine ganze Größenordnung – dies gilt allerdings nur, wenn die Länge des Randes bzw. die Größe der Oberfläche begrenzt ist. Kleine Denkfrage: Wie viele Zellen enthält ein Quadtree, der ein Objekt modelliert, das durch die Hilbert-Moore-Kurve berandet ist?