Fraktale Bild- und Videokodierung

Holger Rauhut[1]
Thomas Riegel[2]
Zentralbereich Technik
Siemens AG
Otto-Hahn-Ring 6
81730 München
Deutschland

18. September 1997

[1]email: HRauhut@compuserve.com
[2]email: Thomas.Riegel@mchp.siemens.de

Inhaltsverzeichnis

Einführung

Die fraktale Bildkodierung ist ein relativ junges Verfahren zur verlustbehafteten Datenkompression von Bildern. 1989 hatte Barnsley zum ersten Mal die Idee, Bilder mit Hilfe von Fraktalen zu kodieren. 1990 wurde dann von A. Jaquin der erste Algorithmus vorgestellt. (Barnsley und Jaquin sind Mitgründer der Firma Iterated Systems und haben ein US-Patent auf dem Verfahren [10, 11].)

Barnsley hatte anfangs enorme Kompressionsraten bis zu 1:10000 proklamiert. Solch hohe Werte wurden dann zwar nicht erreicht, aber die fraktale Kompression kann durchaus mit konventionellen Verfahren, wie zum Beispiel JPEG konkurrieren, wenn sie nicht sogar besser ist.

Die Idee, Bilder mit Hilfe von Fraktalen zu kodieren, beruht auf der Festellung von Benoit Mandelbrot anfang der 80er Jahre [42], daß die Natur sehr gut mit Hilfe von fraktaler Geometrie beschrieben werden kann.

Bei der fraktalen Bildkompression wird versucht, ein Bild als Fraktal zu interpretieren, das mit Hilfe bestimmter Abbildungen erzeugt wurde. Ein solches Fraktal enthält im allgemeinen selbstähnliche Strukturen. Durch Auffinden solcher Selbstähnlichkeiten in einem gegebenen Bild ist es dann möglich, Abbildungen zu bestimmen, durch die ein (fraktales) Bild erzeugt werden kann, das dem zu kodierenden ähnlich sieht. Zur Speicherung eines solchen Bildes ist dann nur noetig, die Koeffizienten zu speichern, die die Abbildungen beschreiben. Je nachdem, wieviele dies sind, wird mehr oder weniger Speicherplatz eingespart.

Fraktale haben die Eigenschaft, daß sie bei weiterer Vergrößerung immer weitere Strukturen zeigen. Diese Eigenschaft hat auch das fraktal kodierte Bild. Es wird auflösungsunabhängig kodiert. Dies bedeutet, daß es in jeder beliebigen Auflösung dekodiert werden kann, auch in höherer Auflösung als das Originalbild. Dabei entstehen natürlich Strukturen, die vorher nicht vorhanden waren. Diese Strukturen passen sich in der Regel aber sehr gut im Bild ein. So werden zum Beispiel Kanten geglättet. Das entstehende Bild sieht in der Regel sogar schöner aus als das vergrößerte Originalbild, bei dem sich die einzelnen Pixel mitvergrößern.

Die fraktale Kompression hat den Nachteil, daß sie sehr rechenzeitaufwendig ist. So lagen die Kompressionszeiten anfangs bei mehreren Stunden. Es wurden aber viele Anstrengungen unternommen, das Verfahren zu beschleunigen, so daß mittlerweile ein Bild in unter einer Minute kodiert werden kann. Eine Variation, die Bath Fractal Transform, ist sogar schneller als JPEG.

Die fraktalen Bildkodierung läßt sich auch auf die Kodierung von Bildsequenzen erweitern. Hierzu sind mehrere Ansätze vorgeschlagen worden, allerdings ist die Forschung auf diesem Gebiet noch nicht weit fortgeschritten. Bisher sind wenig Ergebnisse vorhanden.

Übersicht

Dieser Bericht versucht eine Einführung und Übersicht zu fraktalen Bild- und Videokodiertechniken zu geben. Es werden neuere Entwicklungen beschrieben, die sehr vielversprechend sind. Es wird kein Anspruch auf Vollständigkeit erhoben, ein großer Teil der aktuellen Entwicklung ist aber enthalten.

Der Bericht ist folgendermaßen gegliedert. Kapitel 1 behandelt die fraktale Standbildkodierung und Kapitel 2 die fraktale Videokodierung.

Zunächst wird eine Einführung in die Grundlagen fraktaler Kodierung gegeben. Dann werden verschiedene Kodierungsmethoden beschrieben und Details, wie Quantisierung und Wahl des sogennanten Domain-pools. Danach folgt eine Vorstellung von verschiedenen Verfahren, mit denen die Kodierung beschleunigt werden kann und außerdem ein Abschnitt zur Dekodierung.

Es werden dann zwei Varationen der fraktalen Kodierung vorgestellt: die sogenannte Bath Fractal Transform und die kombinierte Fraktal und DCT-Kodierung (JPEG). Schließlich folgt ein Vergleich der verschiedenen Kompressionstechniken. Hierzu habe ich auch das Programm Fractal Imager von Iterated Systems verwendet.

In Kapitel 2 werden verschiedene Ansätze zur Videokodierung beschrieben. Allerdings sind diese noch nicht sehr weit entwickelt, und bisher wurden auch noch keine Vergleiche der Verfahren untereinander und mit JPEG angestellt. Da die Videokompressionssoftware nicht zur Verfügung stand, konnten auch keine direkten Vergleiche angestellt werden.

Im Artikel beziehe ich mich manchmal auf eigene Untersuchungen, die ich im Laufe einer Jugend-forscht-Arbeit gemacht habe. Ich habe während dieser Arbeit einen sogenannten HV-Kodierer mit einigen Geschwindigkeitsoptimierungen implementiert.

Auf folgenden Internet-Seiten befindet sich interessantes Material zur fraktalen Bildkompression:

- **http://inls3.ucsd.edu/y/Fractals:** Die Internet-Seite von Y. Fisher bietet viel Literatur, Software, Informationen über aktuelle Konferenzen und Links zu anderen interessanten Seiten
- **ftp://ftp.informatik.uni-freiburg.de/documents/papers/fractal:** Auf dieser ftp-Site liegen über 100 Artikel zu fraktaler Bildkompression.
- **http://dmsun4.bath.ac.uk/papers.html:** Literatur zur Bath Fractal Transform
- **http://www.iterated.com:** Homepage von Iterated Systems

Eine schöne Einführung in die Fraktale Bildkompression findet sich in [25] und eine Übersicht in [32].

Kapitel 1

Fraktale Bildkodierung

1.1 Grundlagen der fraktalen Bildkodierung

1.1.1 Iterierte Funktionensysteme

Iterierte Funktionensysteme (IFS) bilden die Grundlage für die fraktale Bildkodierung. Um sie sinnvoll in einem Algorithmus anzuwenden, muß man sie allerdings noch erweitern.

Ein IFS besteht aus einem Satz von n affinen Abbildungen in der Ebene $\omega_1, \omega_2, \ldots, \omega_n$, wobei sich eine Abbildung schreiben läßt als

$$\omega_i \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix} \tag{1.1}$$

Ein solches System wird angewendet, indem man jede einzelne Abbildung anwendet, und dann die Bilder vereinigt. Formell läßt sich dies schreiben als

$$W(X) = \bigcup_{i=1}^{n} \omega_i(X) \tag{1.2}$$

wobei X eine Menge von Punkten in der Ebene ist. W wird als Huntchinson-Operator bezeichnet.

Ein solches System wird nun iteriert, indem man mit einem Startbild X_0 anfängt und wiederholt W auf dieses Startbild anwendet. Man erhält so eine Folge von Punktmengen in der Ebene:

$$X_0, X_1 = W(X_0), X_2 = W(W(X_0)), \ldots, X_k = W^{\circ k}(X_0), \ldots$$

$W^{\circ k}$ bezeichnet dabei k-malige Anwendung von W.

Unter der Voraussetzung, daß W kontrahierend ist, konvergiert diese Folge gegen eine Grenzmenge X_∞, die unabhängig von X_0 ist. Diese Grenzmenge wird auch als Attraktor bezeichnet.

Kontraktivität bedeutet dabei folgendes: Sei Y ein vollständiger metrischer Raum, $f : Y \to Y$ eine Abbildung und $d : Y \times Y \to \mathcal{R}$ eine Metrik. Genau dann, wenn gilt:

$$d(f(x), f(y)) \leq s d(x, y) \quad \text{mit} 0 < s < 1 \quad \forall\, x, y \in \mathcal{R} \tag{1.3}$$

nennt man f kontrahierend bezüglich d. s wird als Kontraktivität von f bezeichnet. Um zu definieren, was Kontraktivität von W bedeutet, muß man eine Metrik für Punktmengen in der Ebene einführen. Dies führt zu der sogenannten Hausdorff-Metrik [25, S. 31] Da diese Metrik mehr von theoretischem Interesse ist, werde ich hier nicht weiter darauf eingehen.

Man kann zeigen, daß W kontrahierend ist, wenn jede einzelne Abbildung ω_i kontrahierend ist, also jede beliebige Strecke unter Wirkung von ω_i verkürzt wird.

Falls die Voraussetzung der Kontraktivität für W erfüllt ist, gilt folgende Beziehung:

$$X_\infty = W(X_\infty) = \lim_{k\to\infty} X_k = \lim_{k\to\infty} W^{\circ k}(X_0) \quad \forall\, X_0 \tag{1.4}$$

Der Attraktor X_∞ von W ist gleichzeitig der Fixpunkt von W. Außerdem ist X_∞ eindeutig durch W festgelegt. Die Iteration konvergiert also immer gegen denselben Attraktor.

Der Attraktor eines IFS ist im allgemeinen ein Fraktal, d.h. er zeigt bei jeder Vergrößerung Strukturen. Außerdem hat er die Eigenschaft der Selbstaffinität, was eine Erweiterung des Begriffes der Selbstähnlichkeit ist. Dies bedeutet, daß Teile des Attraktors zum ganzen Attraktor affin sind, d.h. durch eine lineare Abbildung auf den ganzen Attraktor abgebildet werden können. Der Attraktor besteht also aus verkleinerten und eventuell gedrehten, gespiegelten und/oder verzerrten (deswegen selbstaffin) Abbildungen von sich selbst.

Abb. 1.1 zeigt eine Folge von Iterationen eines IFS, das aus 5 Abbildungen besteht. Als Startbild wird ein Dreieck gewählt. Nach der ersten Iteration erhält man dann 5 Dreiecke, nach der zweiten 25 Dreiecke usw. Man kann gut erkennen, wie diese Iteration sich immer mehr dem Attraktor nähert, der einem Baum ähnelt. An diesem Beispiel wird schon deutlich, daß man mit Hilfe von IFS Strukturen erzeugen kann, die natürlichen Figuren sehr änlich sind, wie eben diesen Baum.

Da der Attraktor nur vom jeweiligen IFS abhängt, läßt sich dieser eindeutig durch die Koeffizienten der Abbildungen des IFS kodieren. Der Attraktor aus dem Beispiel ließe sich so mit $5 * 6 = 30$ Koeffizienten kodieren. Bei großzügigen 4 Byte pro Koeffizient bräuchte man also 120 Byte, um den Attraktor zu kodieren, während das Bild bei einer Auflösung von 640 x 480 Pixel und 1 Bit pro Pixel einen Speicherbedarf von 38400 Byte hätte. Dies entspricht also einem Kompressionsfaktor von 320. Dieser Wert ist allerdings nicht eindeutig, da man den Attraktor auch bei einer anderen Auflösung erzeugen könnte.

1.1.2 Das inverse Problem

Bei der fraktalen Bildkompression versucht man, ein IFS zu finden, so daß der Attraktor dieses IFS ein gegebenes Bild möglichst gut annähert. Dieses Problem wird als das inverse Problem bezeichnet.

Dieses Problem ist direkt nahezu unlösbar. Es läßt sich aber auf ein einfacheres Problem reduzieren. Die theoretische Grundlage für diese Vereinfachung bildet das Collage-Theorem:

$$d(Y, X_f) \leq \frac{1}{1-s} d(Y, f(Y)) \tag{1.5}$$

wobei Y ein beliebiges Bild, f eine Abbildung, X_f den Attraktor bzw. Fixpunkt von f bezeichnet und s die Kontraktivität von f bezeichnet.

Dies bedeutet folgendes: Es genügt, eine kontrahierende Abbildung W zu finden, so daß der Unterschied $d(Y, W(Y))$ zwischen dem gegebenen Bild Y und $W(Y)$ möglichst gering ist. Es soll also gelten $Y \approx W(Y)$. Das Collage-Theorem besagt dann, daß der Unterschied zwischen dem Bild Y und dem Attraktor X_W von W auch gering ist.

Um ein gegebenes Bild Y mit Hilfe eines IFS zu kodieren, schlug Barnsley folgendes Verfahren vor. Man sucht kleinere Teile Z_j des Bildes Y, die Y vollständig überdecken und die durch lineare kontrahierende Abbildungen ω_j des ganzen Bildes möglichst gut approximiert werden:

$$Z_j \approx \omega_j(Y)$$

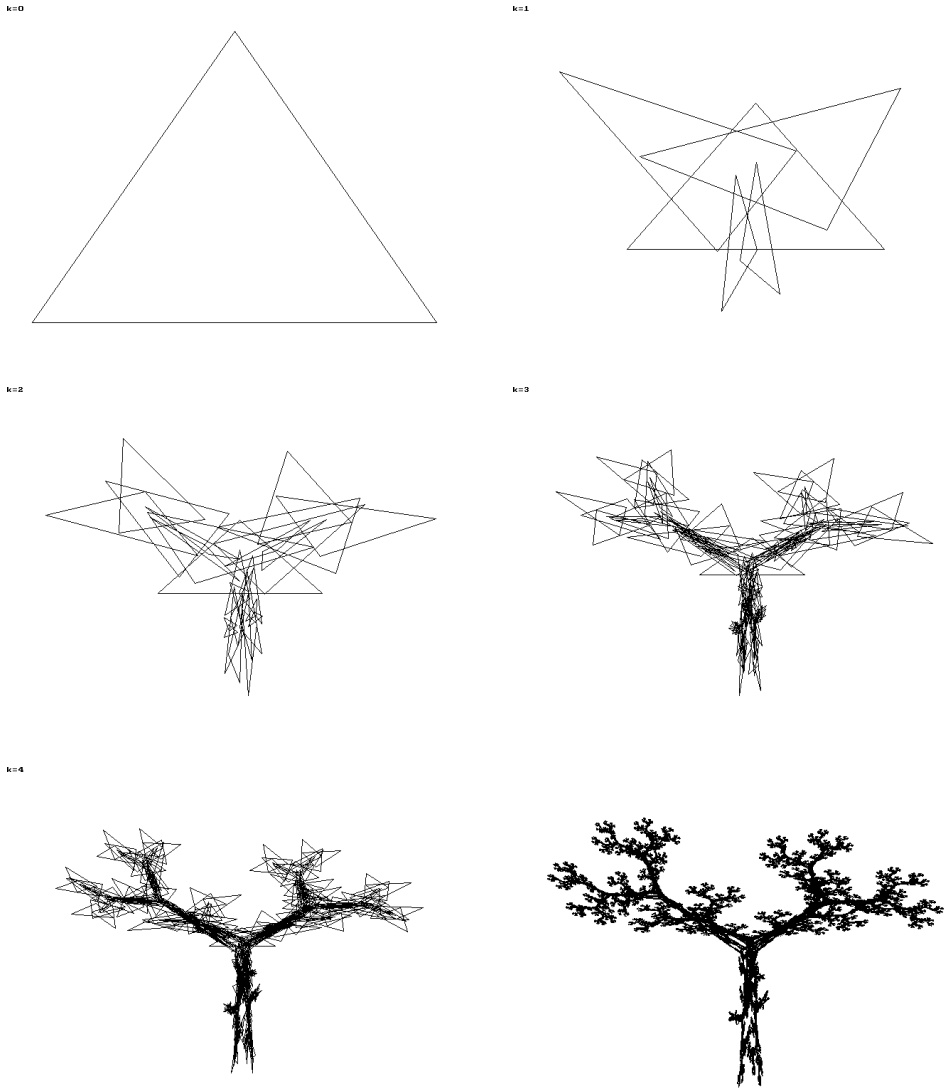

Abbildung 1.1: Ein Dreieck als Startbild, die ersten 4 Iteration und der Attraktor eines IFS

Abbildung 1.2: Ähnlichkeit von Teilbereichen eines Bildes

$W = \bigcup_{j=1}^{n} \omega_j$ bildet dann nach dem Collage-Theorem eine Kodierung des gegebenen Bildes Y.

1.1.3 Iterierte Unterteilungsfunktionensysteme

Für das im letzten Abschnitt beschriebene Verfahren ist es allerdings sehr schwierig, einen automatischen Algorithmus anzugeben, da es nahezu unendlich viele Möglichkeiten gibt, diese Abbildungen bzw. die kleineren Teile des Bildes auszuwählen. Lediglich ein halbautomatisches Verfahren konnte angegeben werden, bei dem interaktiv mit einem Computer das IFS gesucht wird.

Außerdem wird durch ein IFS ein vollständig selbstähnliches Bild kodiert, während die meisten Bilder der Wirklichkeit diese Eigenschaft nicht haben. Und schließlich lassen sich mit IFSen in ihrer Grundform nur Schwarz-Weiss-Bilder kodieren (was aber wohl die kleinste Schwierigkeit ist).

Um die obigen Probleme zu lösen, muß man die IFSe geeignet abändern. Diese Abänderung führt zu den sogenannten Iterierten Unterteilungsfunktionensystemen (IUFS).

Hierzu kann man die Beobachtung machen, daß in den meisten Bildern Teilbereiche vorkommen, die zwar nicht zum ganzen Bild aber dafür zu anderen Teilbereichen ähnlich sind. In Abb. 1.2 sind zwei Paare zueinander ähnlicher Rechtecke in einem Bild markiert. Es dürfte einleuchten, daß Bereich D zu Bereich C ähnlich ist. Außerdem ist Bereich A zu Bereich B ähnlich, und zwar indem man um 180◦ dreht und die Helligkeiten invertiert.

Daraus kann man folgende sinnvolle Abänderung der IFS für die Kodierung von Graustufenbildern (Funktionsweise für Farbbilder später) ableiten.

Ein Bild wird zunächst als Funktion $B : I^2 = [0,1] \times [0,1] \to \mathcal{R}$ betrachtet, wobei $B(x,y)$ die Helligkeit an der Stelle (x,y) bezeichnet. Die Abbildungen ω_i des IFS werden nun um eine Helligkeitstransformation erweitert, so daß sie die folgende

Form haben:

$$\omega_i \begin{pmatrix} x \\ y \\ B(x,y) \end{pmatrix} = \begin{pmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{pmatrix} \begin{pmatrix} x \\ y \\ B(x,y) \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \\ o_i \end{pmatrix} \tag{1.6}$$

s_i ist ein Skalierungsfaktor und o_i ist ein Offset. ω_i ist eine Zusammensetzung aus einer geometrischen Transformation g_i der Form (1.1) und einer Helligkeitsabbildung h_i der Form:

$$h_i(z) = s_i z + o_i \tag{1.7}$$

Die einzelnen Abbildungen ω_i wirken außerdem nicht mehr auf dem ganzen Bild, sondern jeder Abbildung ω_i wird ein Urbildbereich D_i und ein Bildbereich R_i zugeordnet. Die Urbildbereiche D_i werden auch als Domains oder Domain-Blocks und die Bildbereiche R_i als Ranges oder Range-Blocks bezeichnet. Die Ranges müssen das Bild vollständig überdecken und dürfen sich nicht überlappen, damit bei der Anwendung des IUFS jeder Punkt des neuen Bildes $W(B)$ genau einen Helligkeitswert zugeordnet bekommt. Die Domains dürfen beliebig im Bild verteilt sein. Durch diese Abänderung lassen sich dann Ähnlichkeiten zwischen verschiedenen Teilen eines Bildes kodieren.

Wenn $W = \bigcup_{i=1}^{n} \omega_i$ kontrahierend ist, dann gelten wieder alle Ergebnisse, die für IFSe in ihrer Grundform gefunden wurden. Die Iteration konvergiert gegen einen eindeutig bestimmten Attraktor und insbesondere gilt auch das Collage-Theorem (1.5), das für die fraktale Kodierung von großer Wichtigkeit ist. $W = \bigcup_{i=1}^{n} \omega_i$ ist sicher kontrahierend, wenn alle ω_i kontrahierend sind. Umgekehrt kann W aber auch dann kontrahierend sein, wenn nicht alle ω_i kontrahierend sind. Ein Abbildung ω_i ist dabei kontrahierend, wenn die geometrische Abbildung g_i und die Helligkeitsabbildung h_i kontrahierend sind. Die Kontraktion der Helligkeitsabbildung ist sichergestellt, wenn gilt $|s_i| < 1$.

1.2 Funktionsweise der fraktalen Bildkodierung

1.2.1 Grundprinzip

Das Grundprinzip wird an Graustufenbildern erläutert. Eine Möglichkeit, um es auf Farbbilder zu erweitern ist, das Bild jeder Komponente im RGB- oder YUV-Farbraum einzeln zu kodieren.

Zunächst wird ein gegebenes Bild Y möglichst geeignet in Bildbereiche R_i unterteilt, die das Bild vollständig überdecken und sich nicht überlappen. Diese R_i bilden die Ranges der Abbildungen ω_i. Die verschiedenen fraktalen Kompressionsmethoden unterscheiden sich vor allem in der Art der Unterteilung des Bildes. Auf die verschiedenen Unterteilungen wird später eingegangen werden.

Für jedes R_i aus einer geeigneten Auswahl $\mathcal{D}$ (dem sogenannten Domain-pool) wird ein Domain D_i zusammen mit einer kontrahierenden Abbildung ω_i gesucht, so daß die Pixel von R_i möglichst gut durch die Pixel approximiert werden, die durch Anwendung von ω_i auf die Pixel von D_i entstehen. Damit die geometrische Transformation von ω_i kontrahierend ist, wählt man die Domains größer als das zugehörige Range R_i (in der Regel werden die Seitenlängen von D_i doppelt so groß wie die von R_i gewählt). Außerdem läßt man nur Werte von s_i zu für die gilt $|s_i| \leq s_{max}$. s_{max} sollte dabei nicht viel größer als 1 sein. Optimale Werte für s_{max} liegen zwischen 1,2 und 1,5 (siehe auch Kapitel 1.5).

Für die geometrische Transformation, die auch als Isometrie bezeichnet wird, kommen in der Regel nicht sehr viele Möglichkeiten in Frage. Wenn die Ranges quadratisch sind, gibt es zum Beispiel nur 8 mögliche Transformationen, da es nur 8 Möglichkeiten gibt, ein Quadrat auf ein Quadrat abzubilden.

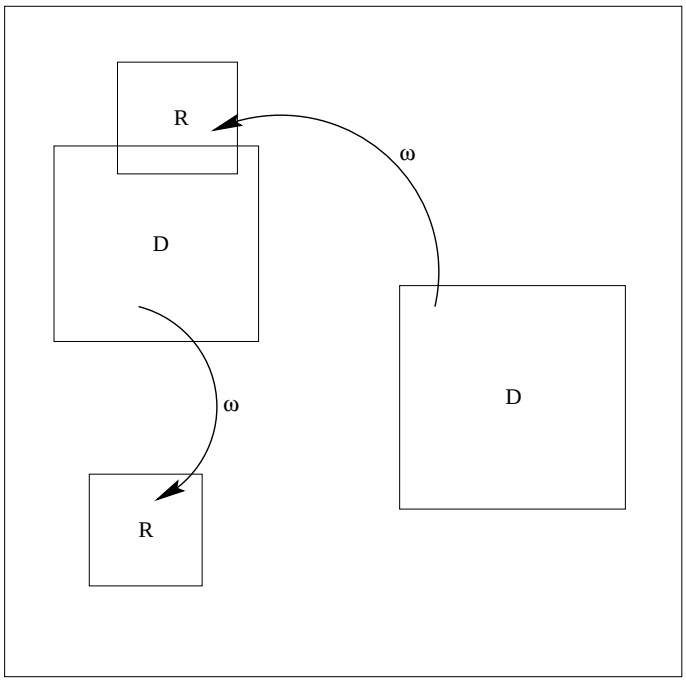

Abbildung 1.3: Prinzip der Iterierten Unterteilungsfunktionensysteme

Nachdem man für jedes R_i ein geeignetes D_i zusammen mit einer Abbildung ω_i gefunden hat, hat man damit eine Kodierung erhalten, die das Bild Y approximiert. Um diese Kodierung zu speichern, muß die Art der Unterteilung des Bildes in Ranges R_i, für jedes R_i das zugehörige D_i, die Art der geometrischen Transformation (Isometrie) von R_i auf D_i und die Parameter s_i und o_i gespeichert werden.

Zur Dekodierung wird mit einem beliebigen Startbild angefangen, zum Beispiel einem schwarzen Bild oder einfach mit dem, was sich zufällig im Speicher befindet, und dann werden wiederholt die Transformationen der Kodierung auf dieses Bild angewendet. Man erhält eine Folge von Bildern, die relativ schnell konvergiert. In der Regel kann man die Iteration nach 5-7 Schritten abbrechen.

1.2.2 Bestimmung der optimalen Abbildung

Ein Range R beinhalte n Pixel. R läßt sich dann als Vektor r der Dimension n schreiben, dessen Komponenten die einzelnen Helligkeitswerte darstellen. Nun sei r' der Vektor, der entsteht, wenn man die geometrische Abbildung g und die Helligkeitsabbildung h auf die Pixel von D anwendet. g bewirkt zunächst eine Reduzierung der Anzahl der Pixel von D auf die von R. (Wenn die Seitenlängen von D doppelt so groß wie die von R sind, kann dies durch einfache Mittelung von 2x2-Pixelblöcken geschehen.) Anschließend werden die Pixel durch g umgeordnet. Wenn g zum Beispiel eine Drehung um 180° ist, wird die Reihenfolge der Pixel genau umgedreht. Der durch diese Prozedur entstandene Vektor sei mit d bezeichnet.

Die Helligkeitsabbildung läßt sich dann schreiben als:

$$r' = sd + o \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \tag{1.8}$$

Um nun die beste Helligkeitsabbildung bzw. die optimalen Parameter s und o zu bestimmen, ist die Größe

$$\|r' - r\| = \|sd + o \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - r\| \tag{1.9}$$

zu minimieren, wobei $\|\cdot\|$ eine geeignet gewählte Norm ist.

In den meisten Implementationen wird hierzu die L_2-Norm verwendet, die definiert ist als

$$\|v\|_2 := \sqrt{\sum_{i=1}^{n} v_i^2} \tag{1.10}$$

Um (1.9) mit der L_2-Norm zu minimieren, ist also die folgenden Größe zu minimieren:

$$T(s,o) = \|r' - r\|^2 = \sum_{i=1}^{n} (sd_i + o - r_i)^2 \tag{1.11}$$

Man erhält daraus folgende Formeln für s und o:

$$s = \frac{n \sum_{i=1}^{n} d_i r_i - \sum_{i=1}^{n} d_i \sum_{i=1}^{n} r_i}{n \sum_{i=1}^{n} d_i^2 - (\sum_{i=1}^{n} d_i)^2} \tag{1.12}$$

$$o = \frac{1}{n}(\sum_{i=1}^{n} r_i - s \sum_{i=1}^{n} d_i) \tag{1.13}$$

Wenn man diese Werte wieder in (1.9) einsetzt, erhält man einen Qualitätswert für die Approximation.

$$T = \sum_{i=1}^{n} r_i^2 + s(s \sum_{i=1}^{n} d_i^2 - 2 \sum_{i=1}^{n} d_i r_i + 2 \sum_{i=1}^{n} d_i) + o(no - 2 \sum_{i=1}^{n} r_i) \tag{1.14}$$

Je niedriger dieser Wert ist, desto besser läßt sich R durch D über eine lineare Transformation approximieren bzw. desto ähnlicher sind R und D. T ist der mittlere quadratische Approximationsfehler.

Um die beste Abbildung für ein bestimmtes Range R zu finden, ist also folgendes Verfahren durchzuführen. Für jedes D aus dem Domain-pool $\mathcal{D}$ und für jede zugelassene Isometrie g, die D auf R abbildet, sind das optimale s und o mit Hilfe von (1.12) und (1.13) zu bestimmen und der Qualitätswert T nach (1.14). Die Kombination (D, g), für die T am geringsten ist, wird zusammen mit den optimalen Werten s, o für diese Kombination als Kodierung des zugehörigen Ranges R verwendet, wobei zusätzlich noch gelten muß: $|s| \leq s_{max}$. Diese Prozedur ist für alle R aus der Unterteilung des Bildes durchzuführen.

1.2.3 Beispiel eines primitiven Kodierers

Um sich das Verfahren etwas besser vorstellen zu können, folgt die Beschreibung eines einfachen Kodierers.

Als Ausgangsbilder für diesen Kodierer dienen Graustufenbilder der Größe 256 × 256 Pixel mit 8 Bit pro Pixel.

Das Bild wird zunächst in 1024 Quadrate der Größe 8 × 8 Pixel eingeteilt. Diese Quadrate sind die Range-Blocks R_i. Zum Domain-pool $\mathcal{D}$ gehören alle möglichen Quadrate der Größe 16 × 16 Pixel. Die Seitenlängen der Domain-Blocks D_i sind damit doppelt so groß wie die der R_i und die Kontraktivitätsbedingung der geometrischen Abbildung g ist also erfüllt. Insgesamt gibt es 241 × 241 = 58081

$(= (256 - 16 + 1) \times (256 - 16 + 1))$ Domain-Blocks. Es werden alle 8 Möglichkeiten zugelassen, D_i auf R_i abzubilden (4 Drehungen und diese 4 Drehungen verknüpft mit einer Achsenspiegelung).

Nun wird für jedes R nach dem in 1.2.2 beschriebenen Verfahren das beste D zusammen mit der besten geometrischen Abbildung g gesucht und gleichzeitig die optimalen Parameter s und o bestimmt nach (1.12, 1.13), wobei man nur solche Paare (D, g) betrachtet, für die $|s| \leq 1$ gilt. Da die Parameter s und o zur Speicherung quantisiert werden müssen, ist es sinnvoll, die Quantisierung schon vor der Berechnung von T nach (1.14) durchzuführen, damit der Fehler berechnet wird, der letztendlich wirklich vorliegt.

Um die Kodierung zu speichern, muß für jedes R_i die Position des zugehörigen D_i gespeichert werden. Hierzu werden 16 Bit benötigt, jeweils 8 Bit für die x- und y-Koordinate. 3 Bits werden für die geometrische Transformation gebraucht, s wird mit 7 Bits gespeichert und o mit 5 Bits. Insgesamt werden so 31 Bits für jede Transformation benötigt. Bei 1024 Transformationen sind dies also 31744 Bits, was 3968 Bytes entspricht. Da die Größe des ursprünglichen Bildes 65536 Bytes beträgt, ist der Kompressionsfaktor 16,52. Die Bildqualität, die sich bei diesem Verfahren ergibt, ist sogar überraschend gut in Anbetracht des etwas naiven Algorithmus (siehe letztes Bild von Abb. 1.4).

Bei diesem Beispiel sieht man schon relativ gut, daß das Verfahren in seiner Grundform sehr rechenzeitaufwendig ist. Es müssen nämlich insgesamt 1024 x 58081 x 8 = 475 799 552 Vergleiche von Quadraten durchgeführt werden.

Da alle Domain-Blocks mit jedem Range-Block verglichen werden müssen, ist es sinnvoll, vor dem Start des Suchalgorithmus eine Tabelle anzulegen, in der für jeden Domain-Block die Summe der Pixel und die Summe der Quadrate der Pixel gespeichert sind (zur Summierung werden die Mittelwerte aus jeweils 2x2 Pixelblöcken verwendet), da diese Werte für jede Berechnung von s, o und T benötigt werden.

Selbst mit dieser Optimierung dauert die Kodierung auf einem Pentium-90-PC etwa 12 Stunden. Die Dekodierung, bei der die Abbildungen auf ein beliebiges Startbild wiederholt angewendet werde, ist dafür wesentlich schneller. Für die Dekodierung werden 6 oder 7 Iterationen benötigt, bis sich kein merklicher Unterschied mehr bei weiterer Iteration ergibt. Auf einem Pentium-90 dauert dies etwa 5 Sekunden. In Abb. 1.4 kann man die Iterationsfolge einer Dekodierung und das dekodierte Grenzbild sehen.

1.2.4 Parallelen zur Vektorquantisierung

Die fraktale Bildkompression ist vom Prinzip her eine Vektorquantisierung, mit dem Unterschied, daß die Codebook-Vektoren aus dem Bild selbst genommen werden und bei der Dekodierung iterativ wieder erzeugt werden. Man spricht deshalb manchmal auch von Selbstvektorquantisierung.

Das Codebook besteht bei der fraktalen Kodierung aus allen Blöcken, die auf die Dimensionen des Range-Blocks gebracht worden sind durch Pixelmittelung und anschließend transformiert worden sind (durch Drehung, Spiegelung, ...). Die einzelnen Pixelwerte dieser Blöcke sind die Komponenten der Codebook-Vektoren.

Für jeden Range-Block-Vektor R wird dann aus dem Codebook der Vektor D gesucht, der R über eine Tranformation am besten approximiert.

1.3 Unterteilungsmethoden

Vor allem die Einteilung eines Bildes in Ranges bestimmt entscheidend das Verhältnis von Bildqualität zum Kompressionsfaktor. Es dürfte plausibel sein, daß die Bildqualität eines fraktal kodierten Bildes umso höher wird, desto höher die Anzahl der

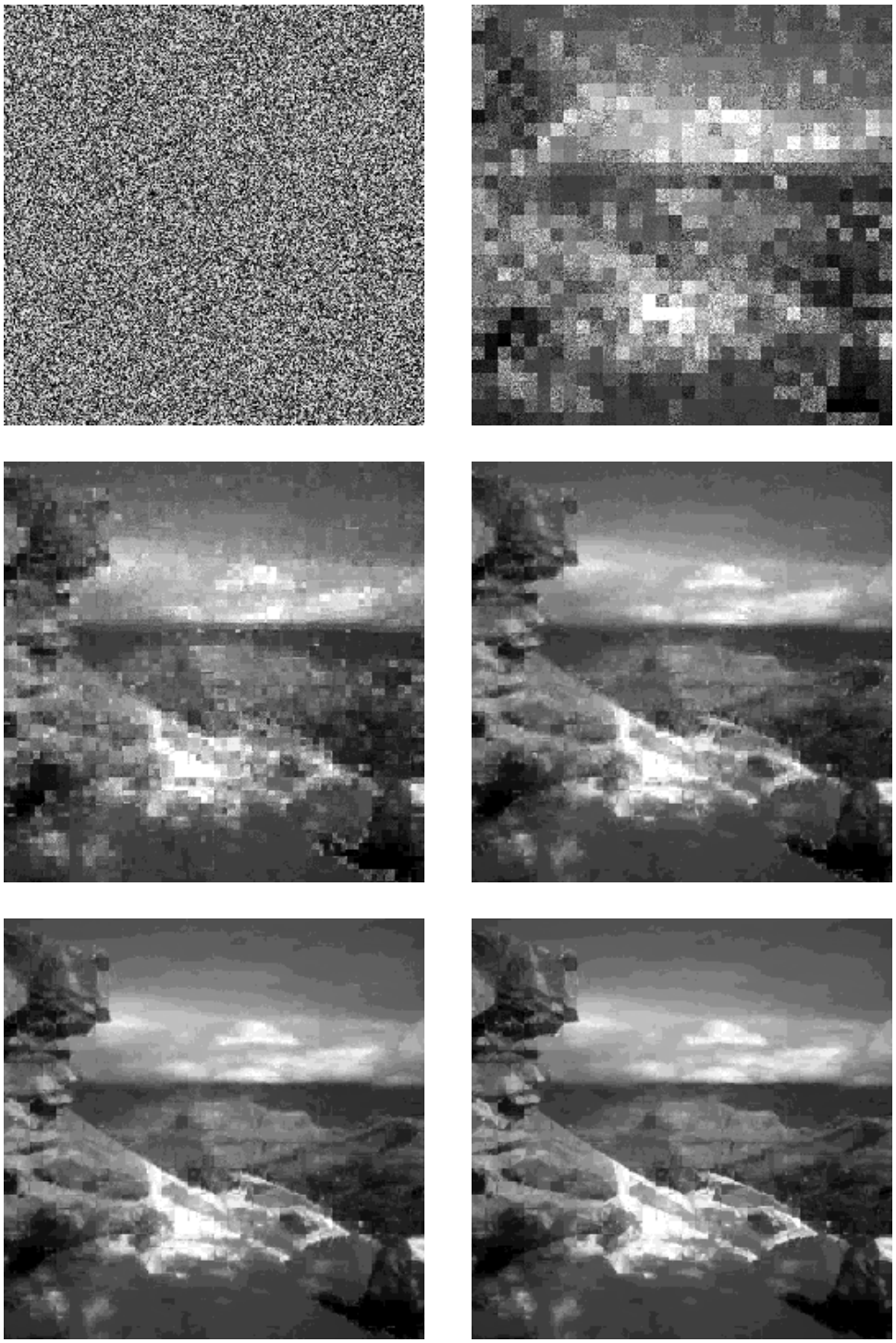

Abbildung 1.4: zufälliges Startbild, die ersten 4 Iterationen der Dekodierung und Endbild

Ranges bzw. Abbildungen der Kodierung ist und je kleiner die einzelnen Ranges sind. (Wenn alle Ranges nur aus einem einzigen Pixel bestehen, ist der Code exakt, allerdings ist der Speicherplatzbedarf dann höher als vorher).

Wenn man eine bestimmte Mindestbildqualität bzw. eine maximale Abweichung vom Originalbild vorgibt, ist es in manchen Bildbereichen sinnvoll im Sinne eines möglichst hohen Kompressionsfaktors, die Ranges nicht zu klein zu wählen, da eventuell auch weniger Ranges, die dann etwas größer sind, diese Bildbereich genau genug kodieren. Dies ist vor allem dann der Fall, wenn ein Bildbereich wenig Struktur aufweist, also zum Beispiel eine einfarbige Fläche beinhaltet. Andererseits ist es in manchen Bildbereichen besser, wenn diese in mehr Ranges unterteilt werden, da diese sonst nicht genau genug kodiert werden können.

Eine gute Unterteilungsmethode muß sich also dem Bild anpassen können. Je nach Bildinhalt, müssen manche Teile des Bildes in mehr und manche in weniger Ranges eingeteilt werden. Dabei muß aber auch bedacht werden, daß ein komplizierteres Verfahren in der Regel zwar bessere Bildqualität liefert, aber die Unterteilung des Bildes selbst mehr Speicherplatz braucht. Außerdem ist die Rechenzeit der Kodierung, die ja ein wesentliches Problem der fraktalen Bildkompression ist, bei einem einfachen Unterteilungsverfahren in der Regel kürzer.

Es wurden verschiedene Methoden vorgeschlagen, ein Bild in Ranges R zu unterteilen:

1. Quadtree-Partition
2. HV-Partition
3. verschiedene Dreicksunterteilungen

Die ersten beiden Methoden und Variationen von diesen Methoden wurden bisher am meisten untersucht. Es gibt zwar noch weitere etwas ausgefallenere Unterteilungen, wie Unterteilung in Polygone und kombinierte Dreiecks-/Vierecks-Unterteilung [21]. Diese seien aber nur der Vollständigkeit wegen erwähnt.

1.3.1 Quadtree-Partition

Die Quadtree-Unterteilung wurde von Y. Fisher vorgestellt [26]. Bei dieser Unterteilungsmethode wird das Bild in verschieden große Quadrate eingeteilt. Die Unterteilung kann dann durch einen Vierbaum (Quadtree) dargestellt werden.

Der Einfachheit sei das zu kodierende Bild Y zunächst quadratisch und seine Seitenlänge sei 2^n, wobei n ganzzahlig ist. Im ersten Schritt der Unterteilung wird das Bild in 4 Quadrate der Seitenlänge 2^{n-1} unterteilt. Im zweiten Schritt der Unterteilung wird jedes der 4 neuen Quadrate wieder in 4 Quadrate der Seitenlänge 2^{n-2} unterteilt. Dieser Vorgang wird solange fortgesetzt bis die Quadrate eine bestimmte vorgegebene Seitenlänge haben. Sinnvoll ist hier 16 oder eventuell auch 32.

Diese Quadrate sind die Range-Blocks R_i. Nun wird für jedes R_i nach dem unter Kapitel 1.2.1 beschriebenen Verfahren das beste D_i aus dem Domain-pool $\mathcal{D}$ zusammen mit einer geeigneten Transformation gesucht. Außerdem wird der Wert

$$\rho = \frac{T}{n} \tag{1.15}$$

berechnet, wobei n die Anzahl der Pixel von R_i ist und T den Approximationsfehler nach (1.14) darstellt. ρ ist der mittlere quadratische Fehler. Falls ρ kleiner als ein vorgegebener Wert ρ_{max} ist, wird die gefundene Transformation als Kodierung für dieses R_i verwendet. Falls ρ größer als ρ_{max} ist, wird R_i weiter in 4 Quadrate unterteilt. Diese Quadrate sind dann die neuen R_i und die oben beschriebene Prozedur wird wieder auf jedes dieser 4 neu erhaltenen Quadrate angewendet.

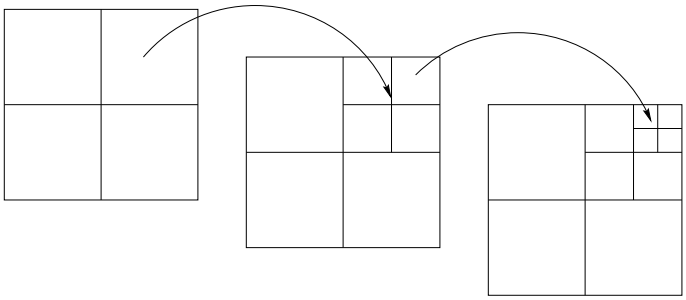

Abbildung 1.5: Quadtree-Unterteilung

Der Parameter ρ_{max} kann zur Steuerung der Qualität des kodierten Bildes bzw. des Kompressionsfaktors verwendet werden.

Die Quadrate werden also immer weiter unterteilt, wenn sie nicht genau genug kodiert werden können, bis eine bestimmte Minimalseitenlänge erreicht ist. Diese wird in der Regel auf 4 festgesetzt. Durch diesen Algorithmus wird die Unterteilung des Bildes dem Bildinhalt angepaßt.

Abb. 1.5 zeigt das Prinzip der Quadtree-unterteilung.

Der Domain-pool besteht aus Quadraten, deren Seitenlängen doppelt so groß wie die der Range-Block-Quadrate sind. Dies bedeutet, es gibt für jede Größe von Range Blocks einen eigenen Domain-pool. Wie dieser genau ausgewählt wird, wird in Kapitel 1.4 diskutiert.

Die Kodierung wird folgendermaßen gespeichert: Für die Unterteilung wird jeweils ein Bit verwendet, um anzuzeigen, ob das jeweilige Quadrat weiter unterteilt wurde oder nicht. Falls auf jeden Fall unterteilt wird, also wenn das Quadrat größer als die Maximalgröße ist, kann dieses Bit weggelassen werden und wenn die Seitenlänge des Quadrats gleich der Minimalseitenlänge ist, kann es ebenfalls weggelassen werden, da auf keinen Fall unterteilt wird.

Die Anzahl der Bits die benötigt werden, um die Position eines Domain-Blocks D_i zu speichern, hängt von der Größe des Domain-pools $\mathcal{D}$ ab. Dies sind je nach Implementation etwa zwischen 500 und 60000 Domains. Es werden also etwa 10 bis 16 Bits für diese Information gebraucht. Für die geometrische Transformation werden 3 Bits benötigt. Der Parameter s wird mit 4 oder 5 Bits und der Parameter o mit 6 oder 7 Bits quantisiert und beide Parameter werden danach entropiekodiert, also mit einem verlustlosen Verfahren komprimiert. Die Quantisierung und Entropiekodierung wird noch genauer in Kapitel1.5 beschrieben.

Um Rechenzeit zu sparen, wird vor Start des Unterteilungs- und Suchalgorithmus, eine Tabelle mit Informationen über die Domain-Blocks angelegt. In dieser Tabelle werden die Summen der Pixel und die Summen der Quadrate der Pixel von allen Domain-Blocks gespeichert, die ja immer für die Berechnung von s, o und T benötigt werden.

1.3.2 HV-Partition

Die HV-Partition, von Yuval Fisher und S. Menlove vorgestellt wurde [27], entstand aus der Motivation, die Unterteilung noch besser dem Bildinhalt anzupassen, als es mit der Quadtree-Partition möglich ist.

Bei der HV-Partition wird das Bild in Rechtecke eingeteilt. Dabei wird folgendermaßen vorgegangen. Als Startpunkt der Unterteilung wird das ganze Bild verwendet. Dieses wird nun an einer geeignet gewählten Position horizontal oder vertikal

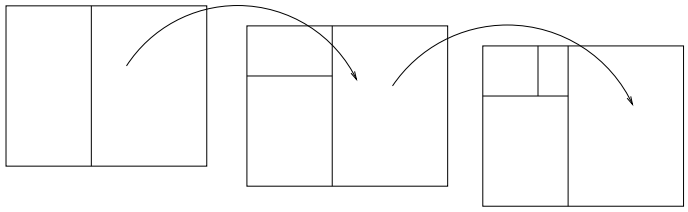

Abbildung 1.6: Die ersten 3 Schritte einer HV-Partition

geteilt (deswegen der Name HV-Partition: H=Horizontal, V=Vertikal). Man erhält so zwei neue Rechtecke. Auf jedes der neu erhaltenen Rechtecke, wird dieser Vorgang weiter angewendet. Man erhält durch rekursive Fortsetzung dieses Prozesses eine Unterteilung des Bildes in Rechtecke. Diese Unterteilung kann als Binärbaum dargestellt werden. in dieser Rangfolge Zur Festlegung der Unterteilungsposition i relativ zum Beginn des Rechtecks, wird ein Wert t_i für jede Spalte bzw. Zeile berechnet:

$$t_i = \frac{\min\{i, N-i-1\}}{N-1}|S_i - S_{i-1}| \tag{1.16}$$

Dabei ist S_i die Summe der Pixelwerte in der i-ten Zeile bzw. Spalte und N die Breite bzw. Höhe eines Rechtecs. Die Stelle i, an der t_i maximal ist, wird als Unterteilungsposition verwendet. t_i ist ein Wert, der den Kontrast zweier benachbarter Pixelzeilen bzw. Spalten angibt gewichtet durch die Position i im Rechteck. Wenn zum Beispiel das Rechteck durch eine horizontale Linie in einen hellen und dunklen Bereich unterteilt wird, wird genau an dieser Grenze unterteilt.

Es gibt bei der HV-Unterteilung zwei Ansätze zu entscheiden, ob ein Rechteck horizontal oder vertikal unterteilt werden soll. Eine Möglichkeit ist, horizontal zu unterteilen, wenn die Länge eines Rechtecks kleiner als die Breite ist, und vertikal im anderen Fall. Dies hat später bei der Speicherung den Vorteil, daß man kein Bit benötigt, um diese Information zu kodieren.

Die andere Möglichkeit ist, die Zeile i zu bestimmen für die t_i maximal wird und die Spalte j für die t_j maximal wird. Horizontal wird dann geteilt, wenn $t_i > t_j$, im anderen Fall wird vertikal geteilt. Diese Methode hat den Vorteil, daß die Unterteilung eventuell besser dem Bildinhalt angepaßt ist.

Es hat sich herausgestellt, daß die erste Methode geringfügig besser als die zweite ist.

Wenn die Seitenlänge eines Rechteck, das durch diesen rekursiven Unterteilungsprozeß entstanden ist, kleiner als ein vorgegebener Wert geworden ist (sinnvoll für diesen Wert ist 20 bis 30), wird der Domain-pool $\mathcal{D}$ (Auswahl von $\mathcal{D}$ siehe Kapitel 1.4) nach dem optimalen D_i zusammen mit einer Transformation g_i abgesucht nach dem unter Kapitel 1.2.1 beschriebenen Verfahren. Wenn der Wert T nach (1.14 für dieses D_i kleiner als ein vorzugebender Wert T_{max} ist, wird das Tripel (R_i, D_i, g_i) zusammen mit den optimalen Parameter s_i, o_i für die Kodierung verwendet. Falls $T > T_{max}$, wird das Rechteck weiter rekursiv unterteilt. In Abb. 1.6 kann man die ersten Schritte einer HV-Unterteilung sehen und in Abb. 1.7 die vollständige Unterteilung eines Bildes.

Es ist sinnvoll, die Größe der Rechtecke durch eine untere Schranke zu begrenzen. Gute Werte für die minimale Seitenlänge liegen je nach erwünschter Bildqualität zwischen 2 (sehr gute Bildqualität) und etwa 10 (mäßige Qualität bzw. hoher Kompressionsfaktor). Wenn man die Größe beschränkt, braucht man später auch durchschnittlich weniger Bits, um die Lage der Unterteilungsposition zu speichern, da weniger Möglichkeiten dafür zugelassen werden.

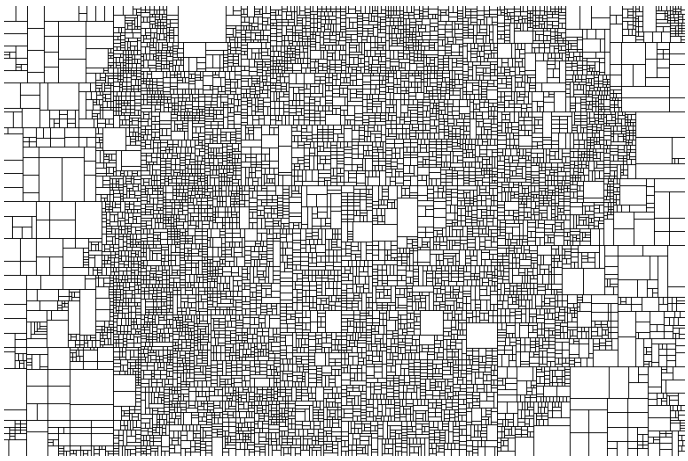

Abbildung 1.7: Ein vollständige HV-Unterteilung eines Bildes

Wie bei der Quadtree-Unterteilung, gibt es für jede verschiedene Rechteckgröße der Ranges einen eigenen Domain-pool. Dieser besteht aus Rechtecken, deren Seitenlängen doppelt so groß wie die von R_i sind. In Kapitel 1.4 wird die Auswahl des Domain-pools diskutiert.

Für die geometrische Abbildung g_i kann man entweder 4 oder 8 Möglichkeiten zulassen. Wenn man nur solche D_i zulassen will, die die gleiche Orientierung haben, wie die zugehörigen R_i, stehen 4 mögliche Abbildungen zur Verfügung (Identität, Spiegelung am Ursprung, Spiegelung an der Mittelsenkrechten und Spiegelung an der Mittelwaagrechten), wenn man außerdem Rechtecke als Domain-Blocks zuläßt, deren Orientierung um 90° gedreht ist, stehen insgesamt 8 Möglichkeiten zur Verfügung.

Um die Unterteilung zu speichern, wird ein Bit verwendet, um anzuzeigen, ob ein Rechteck weiter unterteilt wurde oder nicht. Dieses Bit kann unter den gleichen Bedingungen wie bei der Speicherung einer Quadtree-Kodierung weggelassen werden. Falls nicht weiter unterteilt wurde, folgen die Transformationsparameter: D (Bitanzahl je nach Größe des Domain-pools), g (2 oder 3 Bits, je nachdem, ob 4 oder 8 Möglichkeiten für g zugelassen wurden), s (4 oder 5 Bits) und o (6 oder 7 Bits). Falls weiter unterteilt wurde, folgt eventuell ein Bit, das aussagt, ob horizontal oder vertikal unterteilt wurde (falls die 2. Methode ausgewählt wurde, zu entscheiden, ob horizontal oder vertikal unterteilt wurdel; im anderen Fall ergibt sich diese Information schon aus den Seitenlängen des Rechtecks), und die Unterteilungsposition relativ zum Beginn des Rechtecks. Bei einer Seitenlänge von 8 werden hierfür zum Beispiel 3 Bits benötigt. Falls man eine Mindestseitenlänge angegeben hat, gibt man die Position relativ zum Beginn des Rechtecks plus der Mindestseitenlänge an. Eventuell verringert sich dadurch die Anzahl der benötigten Bits. Wenn im vorigen Beispiel die Mindestseitenlänge 2 ist, ist die erste mögliche Position 2 und die letzte 5, wenn man von 0 an durchnummeriert. Man braucht also nur noch 2 Bits, um diese Information zu kodieren.

Um Rechenzeit zu sparen, führt man den Suchalgorithmus nicht einfach an allen Rechtecken aus, die nacheinander bei der rekursiven Unterteilung entstehen, sondern ändert die Reihenfolge der Ranges R_i, für die der Suchalgorithmus angewendet wird. Und zwar kodiert man zunächst die Ranges mit der größten Seitenlänge. Danach kodiert man die R_i mit immer weiter abnehmender maximaler Seitenlänge.

Dies hat den Vorteil, daß die Ranges, die dieselben Seitenlängen haben hintereinander kodiert werden und dadurch die Tabelle, die die Summen der Pixelwerte, die Summen der Quadrate der Pixelwerte der D_i und eventuell noch weitere Informationen (siehe Beschleuingungsverfahren) enthält, nur einmal aufgebaut werden muß. Wenn für alle Ranges, die dieselben Seitenlängen haben, der Suchalgorithmus durchgeführt wurde, kann die Tabelle gelöscht werden und die Tabelle für den Domain-pool der nächsten Größe aufgebaut werden. (Um eine Tabelle für alle möglichen Domain-pools gleichzeitig im Speicher zu halten, reicht der Speicherplatz der meisten Computer wohl nicht aus.)

Die HV-Unterteilung ist gegenüber der Quadtree-Unterteilung dem Bildinhalt besser angepaßt, da die Grenzen der Range-Blocks oft auch "natürlichen" Grenzen im Bild entsprechen. Allerdings ist der Speicherbedarf zur Kodierung der Unterteilung größer als der der Quadtree-Unterteilung. Außerdem ist die Kodierung deutlich rechenzeitaufwendiger, wie bei der Quadtree-Partitionierung, da wesentlich mehr verschiedene Domain-pools auftreten.

Untersuchungen haben ergeben, daß das Verhältnis von Kompression zu Bildqualität bei der HV-Partitionierung besser als bei der Quadtree-Partitionierung ist (siehe auch Kapitel 1.10).

1.3.3 Dreiecksunterteilung

Die Quadtree und HV-Partition haben den Nachteil, daß Quadrate und Rechtecke zur Unterteilung verwendet werden. Bei hoher Kompression werden dadurch Blockeffekte im dekodierten Bild erzeugt, die dem Auge unangenehm auffallen. Um diesen Nachteil zu umgehen, wurden Unterteilungen vorgeschlagen [20, 21, 51] die Dreiecke für die Unterteilung verwenden. Die Grenzen dieser Dreiecke verlaufen nun im allgemeinen schräg durch das Bild, was dem Auge wesentlich weniger auffällt.

Ich werde zunächst das etwas einfachere Verfahren von Novak [51] wiedergeben.

Einfache Dreiecksunterteilung

Es wird eine etwas abgeänderte affine Transformation verwendet:

$$\omega_i\left(\begin{pmatrix}\tilde{x}\\ \tilde{y}\\ B(\tilde{x},\tilde{y})\end{pmatrix}\right) = \begin{pmatrix}a_{11} & a_{12} & 0\\ a_{21} & a_{22} & 0\\ a_{31} & a_{32} & a_{33}\end{pmatrix}\begin{pmatrix}\tilde{x}\\ \tilde{y}\\ B(\tilde{x},\tilde{y})\end{pmatrix} + \begin{pmatrix}c_1\\ c_2\\ c_3\end{pmatrix}, \tag{1.17}$$

wobei $B(x,y)$ den Pixelwert an der Position (x,y) bezeichnet.

Das Bild wird rekursiv in Dreiecke unterteilt. Zuerst wird das gesamte Bild diagonal geteilt. Danach wird ein Dreieck in 2 jeweils Dreiecke aufgeteilt, indem der neue Eckpunkt von beiden Dreicken auf die Mitte der längsten Seite des alten Dreiecks gesetzt wird. Abb. 1.8 zeigt die ersten Schritte einer solchen Dreiecksunterteilung. Es wird nun solange weiter geteilt, bis eine Seitenlänge S_{max} unterschritten wird. Dann wird aus dem Domain-pool von Domain-Dreiecken, nach dem Dreieck gesucht, das über eine Transformation der Form (1.17) das Range-Dreieck möglichst gut approximiert.

In [51] wird folgende Methode zur Bestimmung der Transformationsparameter angegeben. Seien (x_A, y_A), (x_B, y_B) und (x_C, y_C) die Eckpunkte des Range-Dreiecks und (x_α, y_α), (x_β, y_{beta}) und (x_γ, y_{gamma}) die zugehörigen Eckpunkte des Domain-Dreiecks. Es gibt insgesamt 6 Möglichkeiten, die Zuordnung der Eckpunkte auszuwählen. Für die Transformation ω_i soll dann gelten:

$$\omega_i(x_A, y_A, B(x_A, y_A)) = (x_\alpha, y_\alpha, B(x_\alpha, y_\alpha)) \tag{1.18}$$

$$\omega_i(x_B, y_B, B(x_B, y_B)) = (x_\beta, y_\beta, B(x_\beta, y_\beta)) \tag{1.19}$$

$$\omega_i(x_C, y_C, B(x_C, y_C)) = (x_\gamma, y_\gamma, B(x_\gamma, y_\gamma)) \tag{1.20}$$

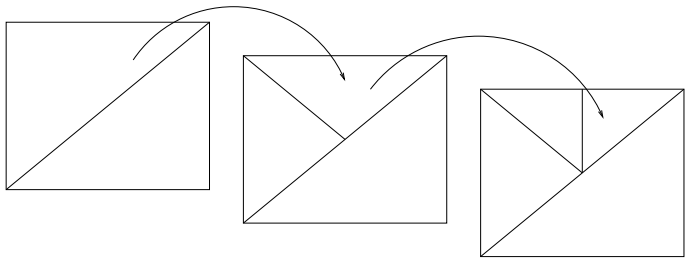

Abbildung 1.8: Die ersten Schritte einer einfachen Dreiecksunterteilung

Die Eckpunkte des domain-Dreiecks werden so auf die Eckpunkte des Range-Dreiecks abgebildet, außerdem werden die Pixelwerte an diesen Eckpunkten genau wiedergegeben. Durch die Bedingungen können 9 Parameter der Abbildung ω_i bestimmt werden. Der zehnte Parameter a_{33} wird so ausgewählt, daß der Approximationsfehler minimal wird. Zur Bestimmung von a_{33} muß also $||r' - r||$ minimiert werden, wobei r den Vektor aus den Pixel des Range-Dreiecks ohne die Eckpunkte darstellt, r' die Approximation von r über eine Transformation der Form (1.17) und $|| \cdot ||$ eine geeignet gewählte Norm darstellt.

Falls der minimale Approximationsfehler (für alle Domain-Dreiecke aus dem Domain-pool) größer als ein festzusetzender Maximalwert T_{max} ist, wird weiter unterteilt, andernfalls wird die gefundene optimale Transformation ω_i für die Kodierung verwendet.

Um eine Abbildung zu speichern, sind folgende Informationen nötig:

- Das Domain-Dreieck, das verwendet wurde (Bitanzahl je nach Größe des Domain-pools)
- Die Isometrie (6 Möglichkeiten, also 3 Bits)
- Die Pixelwerte in den Eckpunkten des Range-Dreiecks (3 Byte)
- Die Pixelwerte in den Eckpunkten des Domain-Dreiecks (3 Byte)
- Der Parameter a_{33} (7 oder 8 Bits)

Um die ganze Kodierung zu speichern, ist außerdem die Unterteilung zu speichern. Hierfür wird jeweils 1 Bit verwendet, um anzuzeigen, ob weiter unterteilt wurde oder nicht.

Mit dem Graustufentestbild "Lenna" im Format 512×512 Pixel wurde bei einem Kompressionsfaktor von 13,52 ein SNR-Wert von 30,1 dB erreicht. Das dekodierte Bild wies kaum Blockeffekte auf. Die Berechnung dauerte 25 Minuten auf einer SUN Sparcstation 2 bei einer Anzahl von 128 Domain-Dreiecken.

Meiner Meinung nach ließe sich mit einer konventionellen Transformation nach (1.6) und einer Minimierung des Approximationsfehlers über alle Pixel des Dreiecks nach (1.12, 1.13, 1.14) eine Verbesserung des Verfahrens erreichen (mit der von Novak vorgeschlagenen Methode werden die Eckpunkte eines Dreiecks genau wiedergegeben und es wird nur über die Pixel im Innern des Dreiecks minimiert).

Unterteilung durch Delaunay Triangulation

Bei dieser von F. Davoine und J.M. Chassery vorgestellten Methode [20] wird eine Delaunay Triangulation [62] zur Unterteilung in Dreiecke verwendet. Mit dieser

Methode läßt sich die Lage und Größe der Dreiecke viel besser dem Bildinhalt anpassen, als mit der unter 1.3.3 beschriebenen einfachen Dreiecksunterteilung. Für die Transformationen werden konventionelle affine Abbildungen der Form (1.6) verwendet.

Um sich dem Bildinhalt anzupassen, sollten in Bereichen des Bildes mit viel Struktur viele Dreiecke sein und in Bereichen mit wenig Struktur sollten weniger Dreiecke sein, die dann natürlich größer sein müssen. Davoine et al. benutzen eine Graphen-Umgebung, um die Manipulation der Dreiecke zu vereinfachen und die Datenstruktur der Unterteilung zu handhaben. Der Unterteilungsalgorithmus besteht aus 2 Schritten: einem Teilungsschritt (Split Step) und einem Vereinigungsschritt (Merge Step).

Es wird mit einer kleinen Anzahl von Punkten angefangen, die regelmäßig im Bild verteilt sind. Diese Punkte stellen die Eckpunkte der Dreiecke dar. Der Teilungsschritt besteht darin, einen Punkt auf den Schwerpunkt eines Dreiecks zu setzen, dessen Pixelwerte nicht homogen verteilt sind. Dabei wird als Kriterium für die Homogenität die Varianz oder der Gradient der Pixelwerte verwendet. Der neue Punkt ist Eckpunkt der hinzukommenden Dreiecke. (Es wird im Unterteilungsstadium übrigens noch gar nicht nach einem passenden Domain-Block gesucht.) Der Prozeß wird solange angewendet, bis die Varianz bzw. der Gradient der Dreiecke einen bestimmten Maximalwert unterschreitet oder die Fläche der Dreiecke eine bestimmte Minimalgröße erreicht hat.

Der Vereinigungsschritt (Merge Step) besteht dann darin, zwei benachbarte Dreiecke zu vereinigen, deren mittlere Helligkeiten etwa gleich sind.

Nach dieser Unterteilung des Bildes wird für jedes Dreieck ein passendes Domain-Dreieck aus einem geeignet gewählten Domain-pool zusammen mit einer Helligkeitsabbildung der Form (1.7) gesucht. Davoine et al. unterteilen nach dem obigen Verfahren das Bild ein zweites Mal, um die Dreiecke des Domain-pools zu erhalten. Meines Erachtens ist dies allerdings unnötig, da für diese Unterteilung wieder Speicherplatz benötigt wird, und die Dreiecke, die entstehen, wenig Struktur aufweisen, was für die Range-Dreiecke zwar gut ist, aber für die Domain-Dreiecke eigentlich nicht erwünscht ist, da diese dann alle sehr ähnlich sind und geringe Varianz auch einfach durch Nullsetzen von s erreicht werden kann. Meiner Meinung nach ist es sinnvoller, den Domain-pool nach einem Schema aus Kapitel 1.4 auswählen.

Um diese Kodierung zu speichern, ist folgende Information nötig:

- die Unterteilung des Bildes in Range-Dreiecke
- für jedes Range-Dreieck die Position des Domain-Dreiecks
- die Orientierung (Isometrie) der Abbildung
- der Skalierungsfaktor s
- der Offset o

Um die Delaunay Triangulation zu speichern, wird für den Teilungsschritt eines Dreiecks 1 Bit verwendet (teilen oder nicht teilen) und 1 Bit für den Vereinigungsschritt (vereinigen oder nicht vereinigen). Diese Folge von Bits kann anschließend noch durch eine Lauflängenkodierung komprimiert werden.

1.4 Wahl des Domain-pool

Es gibt 2 grundsätzliche Strategien, den Domain-pool auszuwählen. In der Literatur findet man dazu seltsamerweise gegensätzliche Ergebnisse, welche nun besser ist.

1. Der Domain-pool besteht aus Domains, die gleichmäßig im ganzen Bild verteilt sind.

2. Der Domain-pool besteht aus Domains, die aus einer kleineren Umgebung des zu kodierenden Range R stammen.

Die zweite Strategie entstand aus der Beobachtung, daß das D, das letztendlich für die Kodierung verwendet wird, oft genau über dem zugehörigen R oder nicht so weit davon entfernt liegt (siehe zum Beispiel [8]. Fisher et al. behaupten allerdings genau das Gegenteil, daß eine solche Beobachtung eben nicht gemacht werden kann [26]. In eigenen Untersuchungen wurde nur die erste Methode ausprobiert, weshalb ich keine Aussage zu diesem Problem machen kann.

Bei beiden Methoden wird ein Gitter über das Bild bzw. über den kleinen Bereich um den Range-Block R gelegt, das eine Gitterweite l hat. Zum Domain-pool $\mathcal{D}$ werden alle Quadrate (Rechtecke, Dreiecke, ...) hinzugenommen, deren linke obere Ecke auf einem Punkt des Gitters liegt. Bei kleinerer Gitterweite l ist $\mathcal{D}$ also größer und bei größerem l kleiner. Man kann durch diesen Parameter also entscheidend die Rechenzeit beeinflußen. Bei kleinerem l braucht man außerdem mehr Bits zur Speicherung der Position von D und bei größerem l weniger.

Untersuchungen (meine eigenen und siehe zum Beispiel haben gezeigt, daß das Verhältnis von Kompression zu Bildqualität umso besser wird je größer $\mathcal{D}$ bzw. je kleiner l ist.

Es ist also schwierig, einen optimalen Parameter l zu finden. Je kleiner l desto länger dauert die Kompression, aber desto besser wird das Verhältnis von Kompression zu Bildqualität und je grösser man l wählt, desto schneller geht die Kompression aber dafür wird das Ergebnis umso schlechter. Man kann allerdings sagen, daß sich bei stark vergrößerter Rechenzeit, das Verhältnis Kompression zu Bildqualität nicht allzusehr verbessert. Sinnvolle Werte für l liegen zwischen 4 und 12.

In [8] wird ein adaptives Verfahren zur Auswahl des Domain-pool vorgeschlagen. Und zwar werden 4 verschiedenen Suchtiefen um die Position des Range-Blocks herum getestet :

1. keine Suche: Es wird der Domain-Block genau über dem zu kodierenden Range-Block verwendet.

2. kleine Suche: Es wird in einem kleinen Bereich um den Range-Block gesucht.

3. große Suche: Es wird in einem relativ großen Bereich um den Range-Block gesucht.

4. volle Suche: Das ganze Bild wird abgesucht.

Für jeden Bereich wird der Domain-Block bestimmt, der den geringsten Approximationsfehler liefert. Schließlich wird der Domain-Block ausgewählt, für den das Verhältnis aus Kompression zu Approximationsfehler am günstigsten ist (wenn zum Beispiel der Domain-Block aus dem ersten Suchbereich genommen wird, braucht man am wenigsten Bits zur Speicherung, allerdings kann es natürlich sein, daß der Range-Block durch einen anderen Domain-Block aus einem größeren Bildbereich besser approximiert wird und die Zunahme der Bildqualität mehr wiegt als die Abnahme des Kompressionsfaktors).

1.5 Quantisierung der Transformationskoeffizienten

Die Parameter s und o der Helligkeitstransformation werden quantisiert, um sie zu speichern. Außerdem muß gelten $|s| \leq s_{max}$ mit einem geeigneten Wert s_{max},

der nicht allzu groß sein sollte, um die Kontraktivität der Vereinigung der Abbildungen sicherzustellen. Damit jede Helligkeitsabbildung kontrahierend wirkt, muß man $s_{max} < 1$ wählen. Man kann aber s_{max} auch etwas größer als 1 wählen, da die Vereinigung von sehr vielen Abbildungen wirkt, von denen nur wenige eine Kontraktivität $|s| \geq 1$ haben. Diese Vereinigung wirkt dann kontrahierend, obwohl nicht alle Abbildungen kontrahierend sind. Werte für s_{max} zwischen 1,2 und 1,5 sind sinnvoll.

Wie schon erwähnt, sollt die Quantisierung noch vor der Berechnung des Abweichungsfehlers T durchgeführt werden, um den Fehler zu berechnen, der nach Quantisierung wirklich vorliegt. Falls gilt $|s| > s_{max}$, wird $s = s_{max}$ bzw. $s = -s_{max}$ gesetzt.

Eine lineare Quantisierung wird dabei angewendet. Es wurden auch Experimente mit nichtlinearen Quantisierungen gemacht, aber dabei haben sich keine Verbesserungen ergeben.

Zunächst wird der Parameter s quantisiert. Der Parameter o wird dann in Abhängigkeit von s quantisiert. Dies ist deswegen sinnvoll, da der Maximalwert und Minimalwert von o vom Wert s abhängen.

Es wurden Experimente gemacht [26], um zu untersuchen, wie die Anzahl der Bits, die für s bzw. o verwendet werden, das Verhältnis Kompression / Bildqualität beeinflussen. Es hat sich dabei herausgestellt, daß bei niedrigem Kompressionsfaktor bzw. hoher Bildqualität 5 Bits für s und 7 Bits für o optimal sind. Bei hohem Kompressionsfaktor und niedrigerer Bildqualität sind dagegen 4 Bits für s und 6 Bits für o optimal. Dies konnte ich auch in eigenen Experimenten bestätigen.

Die Parameter s und o werden vor der Speicherung entropiekodiert, also mit einer verlustlosen Methode komprimiert. Hierbei werden für Werte mit häufigerem Auftreten kürzere Codes verwendet und für Werte mit seltenem Auftreten längere Codes. Ein Huffmann-Kodierer oder arithmetischer Kodierer kann verwendet werden [49, S.27ff]. In meinen Experimenten hat sich gezeigt, daß die Entropiekodierung (durch eine arithmetische Kodierung) allerdings nur noch eine Verbesserung des Kompressionsfaktors um etwa 2 bis 5 Prozent einbringt.

Y. Fisher schreibt in [27], daß zur Entropiekodierung eine feste Häufigkeitsverteilung der o- und s-Werte verwendet werden kann. Dadurch wird der Speicherplatz für die Tabelle zu dieser Verteilung gespart. Die s- und o-Werte sind für verschiedene Bilder nach seinen Experimenten sehr ähnlich verteilt. Um eine gute Verteilungstabelle zu bekommen, sollte man diese über mehrere Bilder und bei mehreren Kompressionsraten mitteln.

Guillermo Ciscar hat sehr viele Entropiekodierverfahren auf Fisher's Quadtree-Code angewendet [18]. Es ergaben sich nur bei 2 Kodierungen Verbesserungen des Kompressionsfaktors um mehr als 5 Prozent. Bei der ersten dieser Kodierungen wurde die Partition des Bildes mit einem arithmetischen Kodierer 1. Ordnung, der Parameter s mit Huffmann, der Offset o und die domain Position mit einem arithmetischen Kodierer 2. Ordnung und die geometrische Transformation mit einem arithmetische Kodierer der Ordnung 0 kodiert. Bei der zweiten dieser Kodierungen wurde die Domain-Position mit "Harri Hirvola's Finite Context Model + Arithmetic Encoder" kodiert (Ciscar hat keine Literaturangabe gemacht, so daß ich leider auch keine angeben kann), die anderen Parameter so wie bei der ersten Kodierung.

1.6 Beschleunigungsverfahren

Die fraktale Kodierung eines Bildes ist sehr rechenzeitaufwendig. Die Kodierzeiten liegen im Minuten bis Stundenbereich bei dem nichtoptimierten Grundverfahren. Der bei Weitem rechenaufwendigste Teil der Kodierung ist dabei die Suche nach einem ähnlichen Block D aus dem Domain-pool $\mathcal{D}$ für einen Range-Block R.

Die Komplexität der Suche für einen Range-Block R verhält sich linear mit der

Anzahl N der Domain-Blocks D in $\mathcal{D}$. N wiederum ist linear mit der Anzahl n der Pixel des Bildes. Die Anzahl der Range-Blocks verhält sich auch linear mit n. Die Gesamtkomplexität der Kodierung ist also $O(n^2)$. Die Kodierzeit wächst also nicht linear, sondern quadratisch mit der Anzahl der Pixel, wenn man den Algorithmus in seiner Grundform betrachtet. Dieses Problem kann man zwar dadurch umgehen, daß man die Größe von $\mathcal{D}$ unabhängig von der Größe des Bildes hält, also die Gitterweite des Gitters auf dem die D_i liegen, umso größer wählt, desto größer das Bild ist. Diese Lösung kann man aber wohl nicht als optimal bezeichnen.

Es wurden relativ viele Verfahren zur Beschleunigung der Kompression vorgeschlagen [7, 16, 17, 33, 40, 55, 56, 57, 58, 59, 60]. Ich werde im folgenden einige näher erläutern, die mir am leistungsfähigesten erscheinen bzw. die ich selber schon getestet habe.

1.6.1 Klassifizierungsverfahren

Ein Ansatz, das Verfahren zu beschleunigen, ist den Domain-pool in verschiedene Klassen einzuteilen, bei der Suche die Klasse des aktuellen Range-Block R zu bestimmen und nur Domain-Blocks aus der derselben Klasse mit R zu vergleichen.

Das Klassifizierungsverfahren, das Y. Fisher in [26] vorschlägt, ist zwar primär für einen Kodierer, der Quadrate als Range-Blocks und Domain-Blocks verwendet, gedacht (Quadtree-Partition). Es läßt sich aber leicht für andere Kodierer modifizieren.

Die KLassifizierung beruht darauf, einen Block in eine Klasse einzuteilen, je nachdem, wie seine Helligkeit auf seine 4 kleineren Unterquadrate verteilt sind. Wenn diese Verteilung für zwei Blöcke gleich ist, werden sie in dieselbe Klasse eingeteilt. Als weiteres Kriterium kann noch die Varianz in den Quadranten betrachtet werden. Der Domain-pool kann so in maximal 72 KLassen eingeteilt werden.

Das Quadrat wird zunächst in 4 Quadranten unterteilt (links oben, rechts oben, links unten, rechts unten). Diese 4 Quadrate werden der Reihe nach durchnummeriert von 1 bis 4. Die Pixelwerte im Quadrant i seien $z_1^{(i)}, \ldots, z_n^{(i)}$. Nun wird für jeden Quadrant

$$A_i = \sum_{j=1}^{n} z_j^{(i)} \tag{1.21}$$

$$V_i = \sum_{j=1}^{n} (z_j^{(i)})^2 - A_i^2 \tag{1.22}$$

berechnet. Es ist nun immer möglich, einen quadratischen Block so zu orientieren (durch Drehung, Spiegelung), daß die A_i auf eine der folgenden 3 Arten geordnet sind (siehe auch Abb. 1.9):

Hauptklasse 1: $A_1 \geq A_2 \geq A_3 \geq A_4$
Hauptklasse 2: $A_1 \geq A_2 \geq A_4 \geq A_3$
Hauptklasse 3: $A_1 \geq A_4 \geq A_2 \geq A_3$

Wenn nun die Hauptklasse des Blocks und damit auch die Orientierung des Quadrats feststeht, können die V_i auf 24 verschiedene Arten geordnet sein. Insgesamt erhält man so 72 verschiedene Klassen. Wenn der Skalierungsfaktor S_i allerdings negativ ist, so ändert sich die Ordnung der A_i und V_i.

In einer Implementation wird vor dem Start des Suchalgorithmus für jeden Domain-Block aus dem Domain-pool die Klasse bestimmt, einmal für positive s_i und einmal für negative s_i, und in diese beiden Klassen wird der Domain-Block

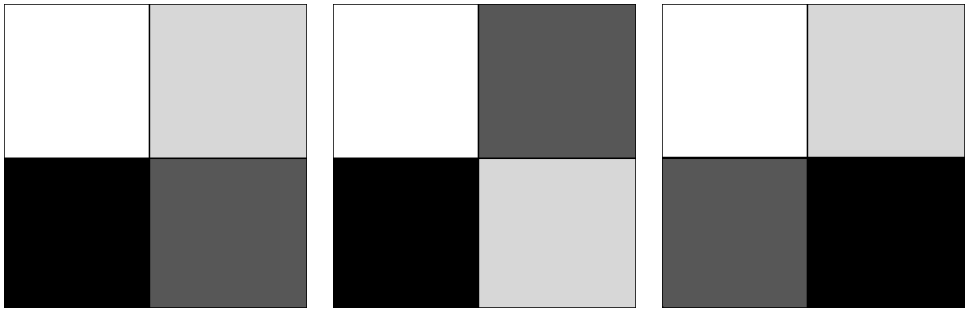

Abbildung 1.9: Ein Quadrat kann immer so orientiert werden, daß die mittleren Helligkeiten seiner 4 Quadranten auf eine der 3 obigen Arten angeordnet sind

eingetragen. Beim Suchalgorithmus wird nun zuerst die Klasse des Range-Blocks bestimmt und dann nur in dieser Klasse nach einem passenden Domain-Block gesucht.

Man hat noch als Option, alle 72 Klassen zuzulassen oder nur die 3 Hauptklassen zu verwenden. Die Bildqualität wird bei letzterem besser sein, allerdings wird die Kompressionszeit deutlich größer werden.

In [26] wird über eine Implementation in einem Quadtree-Coder berichtet. Bei Verwendung von 3 Klassen hat sich eine Beschleunigung etwa um den Faktor 2,5 gegenüber einer vollen Suche ergeben bei einer gleichzeitigen Abnahme des PSNR-Wertes um etwa 0,3 dB. Bei 72 Klassen war die Kodierung etwa um den Faktor 50 schneller, während die Bildqualität um etwa 1,5 dB abnahm gegenüber der vollen Suche.

In eigenen Experimenten habe ich auch ein Klassifizierungsverfahren ausprobiert, bei dem ein Block in 9 Quadranten (3×3) aufgeteilt wird. Es hat sich aber gezeigt, daß die Aufteilung in 4 Teilblöcke effizienter ist.

1.6.2 Nearest-Neighbor-Search

Das Klassifizierungsverfahren bringt zwar einen deutlichen Geschwindigkeitszuwachs der Kodierung, allerdings wird die Komplexität der Kodierung nicht reduziert, sondern nur der Proportionalitätsfaktor in der linearen Komplexität eines einzigen Suchvorgangs wird verkleinert. Dietmar Saupe hat ein Verfahren vorgeschlagen [55, 56, 57, 58, 59], mit dem die Komplexität eines Suchvorgangs auf $O(\log N)$ reduziert wird, und damit die Gesamtkomplexität auf $O(n \log n)$ (N : Größe des Domain-pools, n: Anzahl der Pixel des Bildes). Dieses Verfahren beruht darauf, die sequentielle Suche durch eine multidimensionale Nächste-Nachbarn-Suche (*multidimensional nearest neighbor search*) zu ersetzen. Für dieses Problem gibt es gut untersuchte Algorithmen, die in logarithmischer Zeit arbeiten

Zunächst sei folgende Notation eingeführt: Sei $r \in \mathcal{R}^n$ ein Punkt, der die n Pixel in einem gegebenen Range-Block darstellt. Seien $d^{(1)}, d^{(2)}, d^{(3)}, \ldots, d^{(N)} \in \mathcal{R}^n$ die unterabgetasteten und transformierten Domain-Blocks aus $\mathcal{D}$ mit $|\mathcal{D}| = N$. Sei $E(d^{(i)}, r) = \min_{s,\, o \in \mathcal{R}} \|r - (sd^{(i)} + oe)\|$ der kleinstmögliche Fehler der Approximation von r durch eine affine Transformation von $d^{(i)}$, wobei $\|\cdot\|$ die L_2-Norm und $e \in \mathcal{R}^n$ den Einheitsvektor $e = \frac{1}{\sqrt{n}}(1, \ldots, 1)$ bezeichnet. Die Berechnung der optimalen Parameter s, o und des Fehlers $E(d^{(i)}, r)$ muß für alle $d^{(1)}, d^{(2)}, d^{(3)}, \ldots, d^{(N)}$ ausgeführt werden, um den kleinstmöglichen Fehler $\min_{1 \le i \le N} E(d^{(i)}, r)$ zu erhalten. Diese Prozedur ist sehr rechenzeitaufwendig und muß für viele Punkte r durchgeführt werden. $\langle \cdot, \cdot \rangle$ sei im folgenden das natürliche Skalarprodukt und $\delta(\cdot, \cdot)$ der

euklidische Abstand in $\mathcal{R}^n$.

Seien $\phi : \mathcal{R}^n \to \mathcal{R}^n$ und $\Delta : \mathcal{R}^n \times \mathcal{R}^n \to [0, \sqrt{2}]$ folgendermaßen definiert:

$$\phi(x) = \frac{x - \langle x, e\rangle e}{\|x - \langle x, e\rangle, e\|} \tag{1.23}$$

$$\Delta(d, r) = \min(\delta(\phi(d), \phi(r)), \delta(-\phi(d), \phi(r)) \tag{1.24}$$

ϕ ist die normalisierte Projektion von d auf die Hyperebene, die senkrecht auf e steht.

Es kann nun gezeigt werden, daß gilt [57]:

$$E(d, r) = \min_{s,o \in \mathcal{D}} \|r - (sd + oe)\| = \langle r, \phi(r)\rangle^2 g(\Delta(d, r)) \tag{1.25}$$

$$\text{mit } g(\Delta) = \Delta^2(1 - \Delta^2/4). \tag{1.26}$$

Dies bedeutet, daß der Fehler $E(d, r)$ proportional zu der Funktion g des euklidischen Abstands δ der Projektionen $\phi(d)$ und $\phi(r)$ (oder $-\phi(d)$ und $\phi(r)$) ist. Da g in $[0, \sqrt{2}]$ eine monoton wachsende Funktion ist, kann man die Minimierung von $E(d^{(i)}, r)$ auf die Minimierung von $\Delta(d^{(i)}, r)$ zurückführen ($1 \leq i \leq N$). Diese Minimierung ist äquivalent zu einer Suche nach dem nächsten Nachbarn zu $\phi(r)$ aus der Menge der Vektoren $\pm\phi(d^{(i)}) \in \mathcal{R}^n$.

Für diese multidimensionale Nächste-Nachbarn-Suche gibt es schon gut untersuchte Algorithmen. In [29] wird eine Methode, die sogenannte k-d-Trees (k-dimensional Trees) verwendet, beschrieben und in [4] wird ein optmaler Algorithmus für das sogenannte Approximate-Nearest-Neighbor-Searching vorgestellt. Approximate-Nearest-Neighbor-Searching bedeutet, daß nicht mit absoluter Sicherheit der wirklich nächste Vektor zu einem Suchvektor gefunden wird, dafür ist die Suche wesentlich schneller. Die Verfahren sind recht ähnlich. Beide arbeiten mit einem Binärbaum, der die Vektoren enthält. Das Approximate-Nearest-Neighbor-Searching läßt sich auch mit k-d-Trees implementieren. Das in [4] beschriebene Verfahren sollte allerdings geringfügig schneller sein.

Beide Algorithmen benötigen eine Inititialisierung, bei der eine Datenstruktur aufgebaut wird (beim ersten Algorithmus der k-d-Tree). Dieser Prozess hat die Komplexität $O(N \log N)$. Die Suche nach dem nächsten Vektor zu einem Vektor r hat die Komplexität $O(\log N)$. Da der Suchalgorithmus bei der Fraktalkodierung sehr oft mit dem selben Domain-pool und verschiedenen Ranges R durchgeführt wird, spielt die Zeit zum Aufbau des Baums nur eine geringe Rolle, da dies nur einmal durchgeführt werden muß. Die Leistungsfähigkeit beider Algorithmen verschlechtert sich allerdings stark mit zunehmender Dimension der Vektoren.

Die Implementation in einem Fraktalkodierer sieht nun folgendermaßen aus:

- Zuerst müßen einige praktische Überlegungen angestellt werden, die den Speicherplatzbedarf betreffen. Wenn der Domain-pool zum Beispiel aus Quadraten der Größe 8x8 Pixel besteht, benötigt man 64 Gleitkommazahlen für jeden projezierten Vektor $\phi(d^{(i)})$. Bei 4 Byte pro Zahl und insgesamt 8000 Domains brauchen alle projezierten Vektoren etwa 2 MByte. Man muß also diesen enormen Speicher reduzieren. Hierzu wird ein Domain-Block vor der Projektion auf eine niedrigere Dimension gebracht. Dies geschieht entweder durch Unterabtastung der Domain-Blocks oder durch Anwendung einer 2-dimensionalen diskreten Cosinustransformation (DCT) auf den Domain-Block. Von den DCT-Koeffizienten werden dann nur die ersten n verwendet. Die Koeffizienten der Vektoren werden mit jeweils 1 Byte gespeichert [57]. (Die Verwendung von Integer-Arithmetik hat den Vorteil, daß Vergleiche, die bei der Suche durchgeführt werden müssen, schneller als mit Gleitkomma-Arithmetik sind.)

  Die Verkleinerung der Dimension ist insofern kein Nachteil, da der Suchalgorithmus mit wachsender Dimension sowieso leistungsschwächer wird.

- Durch die Verringerung der Dimension und die Quantisierung der Koeffizienten der Vektoren $\phi(d^{(i)})$ auf 1 Byte ist der durch den Algorithmus gefundene Vektor nicht mehr unbedingt wirklich der nächste zum Vektor r. Deshalb werden gleich die m nächsten Vektoren zu einem Range-Vektor r gesucht. Danach wird mit Hilfe des konventionellen Fraktalkodierverfahrens das beste D aus diesen m Vektoren bestimmt.

- Um Speicherplatz zu sparen, kann man die Hälfte der Vektoren $\pm\phi(d^{(i)})$ im [57]k-d-Tree weglassen, da der Binärbaum aus den Vektoren $-\phi(d^{(i)})$ symetrisch zum Baum aus den Vektoren$+\phi(d^{(i)})$ ist. Dieser Baum muß dann allerdings zweimal durchsucht werden, einmal mit $+\phi(r)$ und einmal mit $-\phi(r)$. Wenn man außerdem noch mehrere geometrische Abbildungen der D_i zuläßt, ist es nicht nötig, auch alle transformierten Vektoren in den Baum aufzunehmen, da man auch umgekehrt den Range-Block transformieren kann und mit jedem transformierten Range-Block den Binärbaum durchsuchen kann.

Meine eigenen Untersuchungen haben gezeigt, daß Werte von 9 bis 16 für die Dimension n optimal sind. Je größer die Anzahl m der nächsten Domain-Blocks ist, die gesucht werden, desto besser wird die Bildqualität. Sinnvolle Werte liegen zwischen $m = 10$ und $m = 30$.

Bei der Suche werden die ersten m $(1+\epsilon)$-ungefähr nächsten Vektoren bestimmt. Dies bedeutet, daß der Abstand eines gefundenen Vektors zum Suchvektor maximal $(1 + \epsilon)$ mal so groß wie der Abstand des wirklich nächsten Vektors zum Suchvektor ist. Arya et al. [4] haben berichtet, daß selbst bei so hohen Werten, wie $\epsilon = 3$ mit 50 Prozent Wahrscheinlichkeit der wirklich nächste Vektor gefunden wird, und der durchschnittliche Abstand des gefundenen Vektors nur 1.05 mal so groß, wie der des wirklich nächsten ist. Die Rechenzeit ist bei großen ϵ dafür wesentlich geringer (etwa Faktor 12 bei $\epsilon = 3$).

In eigenen Untersuchungen hat sich ergeben, daß $\epsilon = 3$ das optimale Verhältnis von Rechenzeit zu Bildqualität liefert. D. Saupe hat ähnliche Erbegnisse bei seinen Untersuchungen erhalten [57].

In meiner Implementation ließ sich durch die Anwendung des oben beschriebenen Verfahrens die Kodierzeit für ein durchschnittlich großes Bild von etwa 1 Stunde auf 10 Minuten verkürzen. Die Rechenzeit wächst außerdem nicht mehr so schnell mit der Anzahl der Pixel eines Bildes.

1.6.3 Vergleich der vorgestellten Verfahren

Ich habe zwei verschiedene Klassifizierungsverfahren und das Approximate- Nearest-Neighbor-Searching-Verfahren in einen HV-Partitionierungs-Algorithmus implementiert.

Es hat sich gezeigt, daß das letztere der beiden Verfahren deutlich besser ist, was nicht unbedingt zu erwarten war, da in einem HV-Kodierer sehr viele verschiedene Range-Block-Größen vorkommen, also auch sehr viele Domain-pools. Die Initialisierungsroutine, die ja sehr zeitintensiv ist (Komplexität $O(\log N)$), muß also sehr oft durchgeführt werden.

Es hat sich jedenfalls gezeigt, daß das Nearest-Neighbor-Searching in jedem Fall besser als das Klassifizierungsverfahren ist. Vor allem bei sehr großen Bildern wird sehr viel an Rechenzeit eingespart.

1.6.4 Kombination der Verfahren

In [57] wurden das Klassifizierungsverfahren und das Nearest-Neighbor-Searching-Verfahren kombiniert. Durch eine solche Kombination ist ein noch größerer Geschwindigkeitszuwachs zu erwarten. Der Domain-Pool wird zunächst in 3 oder 72

Klassen eingeteilt. Für jede Klasse wird nach dem in Kapitel 1.6.2 beschriebenen Verfahren ein Binärbaum aufgebaut. Bei der Suche wird zunächst die Klasse des Range-Blocks und dann in dieser Klasse mit dem Nearest-Neighbor-Searching-Verfahren nach dem optimalen Domain-Block gesucht.

Die Experimente die Saupe et al. durchgeführt haben, bestätigen, daß sich die Kodierung so noch einmal deutlich beschleunigen läßt. Es wird berichtet, daß die Kodierung um den Faktor 1,3 bis 13,3 schneller wie bei Verwendung des konventionellen Klassifizerungsverfahrens ist. Dabei ist der Beschleunigungsfaktor umso größer je größer das Bild ist.

1.7 Dekodierung

1.7.1 Beschleunigung

Die Dekodierung ist zwar schon relativ schnell, kann aber mit Hilfe eines hierarchischen Dekodierers noch beschleunigt werden [6]. Die Funktionsweise ist folgende: Man beginnnt die Iteration bei einer niedrigeren Auflösung als der Originalauflösung. Als Optimum hat sich herausgestellt, wenn man mit einem Bild der Größe $B/8 \times H/8$ anfängt (B, H: Breite bzw. Höhe des Originalbildes). Nun iteriert man bei dieser Auflsung 2 oder 3 Schritte. Danach verdoppelt man mit jedem Dekodierungsschritt die Auflösung, bis man die Originalauflösung erreicht hat. Jetzt braucht man in der Regel nur noch einen einzigen Schritt, bis eine weitere Iteration keine merklichen Unterschiede mehr erzeugt. Abb. 1.10 zeigt eine solche Iteration.

1.7.2 Nachbearbeitung

Nach der Dekodierung sollte das Bild noch nachbearbeitet werden, da vor allem bei höheren Kompressionsraten Blockartifakte im Bild entstehen. Man kann die Ränder der Range-Blocks gut im Bild erkennen.

Um diesen Effekt abzumildern, wird bei großen Range-Blocks ein Smoothing der Pixel am Rand der Range-Blocks durchgeführt [26, 27]. Wenn die Range-Blocks länger als 6 Pixels in der Richtung sind, die betrachtet wird, werden die Pixel am Rand des Range-Blocks gesmoothed. Wenn sie länger als 10 Pixel sind, werden die ersten beiden Pixel, die am nächsten zum Rand liegen gesmoothed.

Die visuelle Qualität vor allem von stark komprimierten Bildern wird dadurch stark verbessert. Den Effekt des Smoothings zeigt Abb. 1.11.

1.7.3 Auflösungsunabhängigkeit

Wie schon erwähnt, wird das Bild auflösungsunabhängig durch die Transformationen kodiert. Man kann also in jeder beliebigen Auflösung dekodieren (Grenzen setzt nur Speicherplatz und Rechenzeit). Wenn man in größerer Auflösung als das Originalbild dekodiert, werden Pixel eingefügt, die natürlich zunächst einmal mit dem ursprünglichen Bild nichts zu tun hatten. Die Pixel passen sich aber in der Regel recht gut dem Bild an, es werden sozusagen Details interpoliert. Wenn man mit dem vergrößerten Originalbild vergleicht, sieht das fraktal kodierte Bild in der Regel sogar besser aus, da sich bei Vergrößerung des Originalbildes die Pixel mitvergrößern. (Das Bild muß natürlich in genügend hoher Qualität kodiert worden sein.) Abb. 1.12 zeigt Ausschnitte aus einem vergrößerten Originalbild und vergrößerten fraktal kodierten Bild.

Abbildung 1.10: beschleunigte Dekodierung


Abbildung 1.11: links: nicht nachbearbeitetes Bild; rechts: nachbearbeitetes Bild

1.8 Die Bath Fractal Transform

Ein anderer vielversprechender Ansatz zur Fraktalkodierung ist die "Bath Fractal Transform"(BFT), die von D.M. Monro et al., tätig an der University of Bath, erstmals vorgestellt wurde [43, 44, 43, 64].

Die BFT ist eine Verallgemeinerung des in Kapitel 1.1 vorgestellten Verfahrens der fraktalen Bildkompression. Die Helligkeitstransformation ist bei der BFT ortsabhängig. Die konventionelle Fraktalkodierung ist eine BFT der Ordnung 0.

Nach Monro hat die BFT gegenüber der konventionellen Fraktalkodierung den Vorteil, daß sie sehr schnell ist, da auf den Suchalgorithmus verzichtet werden kann.

1.8.1 Grundlagen der BFT

Bis auf die Helligkeitstransformation ist die BFT gleich wie die konventionelle Fraktalkodierung. Ein Bild wird in Ranges R_i eingeteilt, und es wird versucht, diese R_i durch transformierte und heruntergefilterte größere Domain-Blocks, die aus dem gleichen Bild genommen werden, durch eine Transformation zu approximieren.

Sei $r(x, y)$ ein Pixel des Range-Blocks, wobei x, y die Koordinaten dieses Pixel relativ zur linken oberen Ecke des Blocks darstellen und $d(x, y)$ der Pixel des unterabgetasteten und eventuell gedrehten oder gespiegelten Domain-Blocks, der $r(x, y)$ über eine Transformation approximieren soll. Diese Transformation hat bei der BFT folgende Form.

$$r(x, y) = v(x, y, d(x, y)) \tag{1.27}$$

Die Transformation ist also im Gegensatz zur konventionellen Fraktalkodierung zusätzlich abhängig vom Ort.

Monro et al. haben Polynome für die Funktion v vorgeschlagen:

$$v(x, y, d(x, y)) = a + bd(x, y) + c_x x + c_y y + d_x x^2 + d_y y^2 + e_x x^3 + e_y y^3 \tag{1.28}$$

Die Kreuzprodukte von x und y werden nicht berücksichtigt.

Im Falle, daß alle Koeffizienten außer a und b Null sind (BFT der Ordnung 0), entspricht die Transformation der konventionellen fraktalen Transformation (der Parameter a entspricht dem Parameter o und b entspricht s).

Für $d_x, d_y, e_x, e_y = 0$ ist die Transformation bilinear (BFT der Ordnung 1), für $e_x, e_y = 0$ biquadratisch (Ordnung 2) und für alle Koeffizienten ungleich Null

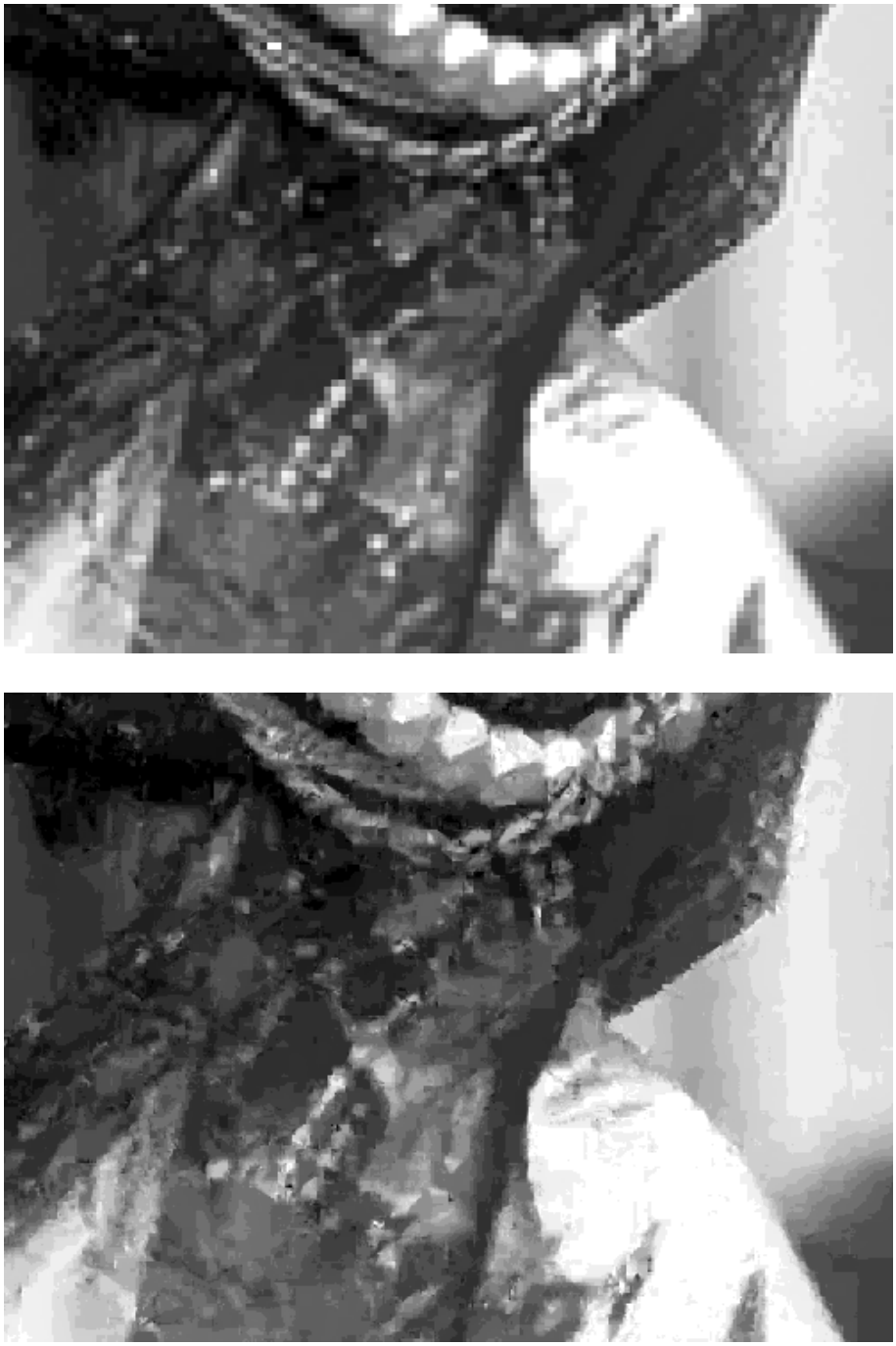

Abbildung 1.12: Ausschnitt aus der Vergrößerung eines Originalbildes (oben) und eines fraktal kodierten Bildes (unten), Vergrößerung jeweils um den Faktor 5

bikubisch (Ordnung 3). Polynome höherer Ordnung sind von D. Monro et. al nicht in Betracht gezogen worden.

Sei nun $d(\cdot,\cdot)$ eine geeignet gewählte Metrik (zum Beispiel die L_2-Metrik). Um die optimalen Paratemer der Transformation zu bestimmen, ist nach dem Collage-Theorem (1.5) der Abstand $d(r, r')$ zu minimieren wobei r' der Vektor ist, der r approximieren soll: $r'(x, y) = v(x, y, d(x, y))$. Dazu ist das Gleichungssystem zu lösen, das durch Nullsetzen der partiellen Ableitungen von $d(r, r')$ nach den einzelnen Koeffizienten entsteht.

$$\frac{\partial d(r,r')}{\partial a} = 0, \frac{\partial d(r,r')}{\partial b} = 0, \frac{\partial d(r,r')}{\partial c_x} = 0, \cdots, \frac{\partial d(r,r')}{\partial e_x} = 0, \qquad (1.29)$$

Für den Fall der L_2-Norm 1.10, und der bilinearen BFT findet sich in [43] das explizite Gleichungssystem, das zu lösen ist.

1.8.2 Kodierung

Monro et al. haben gefunden, daß ein BFT-Kodieralgorithmus, auf die Suche im Domain-pool ganz verzichten kann, oder daß man zumindest nur eine minimale Suche durchführen muß.

Der von Monro et al. vorgeschlagene Algorithmus [48] funktioniert folgendermaßen: Das Bild wird zunächst in Blöcke aufgeteilt, im einfachsten Fall in Quadrate. Nun wird jeder dieser Blöcke noch einmal weitere in 4 kleinere Blöcke aufgeteilt. Diese 4 Blöcke sind die Range-Blocks. Der einzige Domain-Block ist der größere Block. Man betrachtet also nur kontrahierende Abbildungen des größeren Blocks auf die kleineren. Monro et al. haben berichtet, daß man sogar auf Anwendung von Spiegelungen und Drehungen verzichten kann.

Nun werden für jeden Range-Block die Parameter $a, b, c_x, c_y, \ldots$ der Transformation nach (1.29) bestimmt. Diese stellen die fraktale Kodierung des Bildes dar. Um die Parameter zu speichern, wird ein Lloyd-Max-Quantisierer [38] verwendet.

Bei einem solchen Quantisierer wird die Verteilung der Werte bei der Quantisierung berücksichtigt. In Bereichen, in denen die Wahrscheinlichkeit höher ist, daß ein Datenpunkt hineinfällt, wird der Abstand der Quantisierungsstufen kleiner gesetzt. Dadurch wird der mittlere Quantisierungsfehler kleiner.

Durch die Anzahl der Bits, die für die jeweiligen Koeffizienten verwendet werden, läßt sich die Kompression bzw. Bildqualität steuern. Monro et. al haben dabei 4 Qualitätsabstufungen eingeführt [48]:

High :	a, b: 6 Bits;	c_x, c_y, d_x, d_y : 5 Bits
Medium :	a, b : 5 Bits;	c_x, c_y, d_x, d_y : 4 Bits
Low :	a, b : 4 Bits;	c_x, c_y, d_x, d_y : 3 Bits
Poor :	a, b : 3 Bits;	c_x, c_y, d_x, d_y : 2 Bits

Die quantisierten Parameter werden anschließend noch entropiekodiert (siehe auch Kapitel 1.5).

1.8.3 Dekodierung

Zur Dekodierung wird ein von Monro et al. entwickelter Algorithmus verwendet, der sich Accurate Fractal Rendering Algorithm (AFRA) nennt. Dieser Algorithmus ist allerdings noch nicht veröffentlicht.

1.8.4 Ergebnisse

Monro and Woolley haben herausgefunden, daß eine biquadratische BFT (Ordnung 2) das beste Verhältnis aus Kompression und Bildqualität liefert [64]. Sie haben auch Experimente mit einem Algorithmus gemacht, bei dem das Bild nach mehreren Domains durchsucht und außerdem Spiegelungen und Drehungen betrachtet werden. Das Verhältnis Kompression / Bildqualität hat sich dadurch allerdings verschlechtert, da mehr Bits für die Information über die Lage des Domain-Blocks benötigt werden.

Monro et. al haben verschiedene Größen von Domain-Blocks ($6 \times 6, 8 \times 8, 12 \times 12, 16 \times 16und24 \times 24$ Pixelblöcke), die durch kleinere Blöcke approximiert werden, ausprobiert [64]. Dabei hat sich ergeben, daß für höhere Bildqualität bzw. niedrigerer Kompression kleinere Blöcke optimal sind und für höhere Kompression größere Blöcke besser sind, was auch zu erwarten war.

Die BFT bietet gegenüber der fraktalen Kompression den Vorteil, daß auf den Suchalgorithmus verzichtet werden kann. Die Kodierung ist also sehr schnell, so daß sich die BFT auch für Echtzeitanwendungen, wie z.B. Bildtelefon eignet. Monro et al. haben diesen Algorithmus in der Tat auch auf Videokodierung erweitert (siehe Kapitel 2.3).

Bei dem bisherigen Stand der Entwicklung ergeben sich bei der BFT zwar etwas geringere Bildqualitäten als mit JPEG-Kompression [44], allerdings steckt sie noch am Anfang der Entwicklung. Es können also noch Verbesserungen erwartet werden. Die Kodier- und Dekodierzeiten liegen dafür unter denen der JPEG-Kompression. Vergleiche zu anderen Fraktalkodierverfahren wurden noch nicht durchgeführt.

1.9 Kombination von DCT- und Fraktalkodierung

Barthel, Schüttemeyer, Voyé und Noll stellen in [12], [8] und [9] eine Kombination aus DCT (JPEG) und Fraktalkodierung vor. Sie nennen ihre Methode Fractal Transform Coding (FTC). Barthel et al. berichten, daß durch diese Kodiertechnik ein besseres Verhältnis von Kompression zu Bildqualität zu erwarten ist, als mit herkömmlichen Fraktalkodiertechniken erreicht wird.

1.9.1 Grundlagen

Die konventionelle Fraktalkodierung verwendet eine Helligkeitstransformation h der folgenden Form:

$$r = h(d) = sd + o, \tag{1.30}$$

wobei r den Vektor aus den Pixeln des approximierten Range-Blocks darstellt und d den Vektor aus den Pixeln des (geometrisch transformierten) Domain-Blocks.

Barthel et al. schlagen eine Modifikation dieser Helligkeitstransformation vor:

$$r = \lambda(d) = s(d - \mu_d) + 0.5\mu_d + o \tag{1.31}$$

wobei μ_d den Mittelwert des Domain-Blocks d darstellt.

Wenn der Range-Block groß ist und komplexe Strukturen enthält, läßt dieser sich in der Regel nicht sehr gut durch eine solche Helligkeitstransformation approximieren. Konventionelle Fraktalkodierverfahren teilen in so einem Fall den Range-Block auf und suchen für jeden der kleineren Blöcke eine Transformation. Barthel et al. schlagen statt dessen eine Helligkeitstransformation höherer Ordnung vor. Anders als bei der Bath Fractal Transform (siehe Kapitel 1.8) werden keine Polynome 2. oder 3. Ordnung verwendet, sondern es wird eine Helligkeitstransformation im Frequenzraum eingeführt.

Sei D ein (geometrisch transformierter) Domain-Block und R der zu approximierende Range-Block. Die Größe dieser Blöcke sei $N \times N$ Pixel. Sei weiter $\hat{d}$ bzw. $\hat{r}$ der Vektor, der entsteht durch Anwendung der Diskreten Cosinus Transformation (DCT) auf D bzw. R und anschließendes Zick-Zack-Scannen durch die Spektralkoeffizienten (wie bei JPEG-Algorithmus, siehe u.a. [49]).

Die Helligkeitstransformation wird nun gegeben durch:

$$\hat{r}' = \Lambda(\hat{d}) = \begin{pmatrix} s_0 & 0 & \cdots & & 0 \\ 0 & s_1 & & & \\ & & \ddots & & \vdots \\ \vdots & & & s_l & \\ & & & & 0 \\ 0 & & \cdots & 0 & s_{N^2-1} \end{pmatrix} \hat{d} + \begin{pmatrix} o_0 \\ o_1 \\ \vdots \\ o_l \\ \vdots \\ o_{N^2-1} \end{pmatrix}, \tag{1.32}$$

wobei $\hat{r}$ durch $\hat{r}'$ approximiert werden soll.

Diese Darstellung ist natürlich nicht geeignet, Kompression zu erreichen, da N^2 Pixel durch $2N^2$ Koeffizienten kodiert werden. Es muß also eine einfachere Darstellung verwendet werden.

Barthel et al. schlagen folgendes Kodierverfahren als Vereinigung von Fraktalkodierung und Transformationskodierung vor. Der größte Teil des Frequenzspektrums eines Range-Blocks wird fraktal kodiert, d.h. es wird $s_i = s$ gesetzt für fast alle i. Nur die Frequenzkoeffizienten, die nicht genau genug approximiert werden können, werden individuell kodiert, so wie bei der Transformationskodierung. Der Parameter s_0 wird immer auf 0,5 gesetzt, um Kontraktivität sicherzustellen.

Können zum Beispiel die Koeffizienten $\hat{r}_1$ und $\hat{r}_3$ nicht genau genug durch die fraktale Transformation kodiert werden, hat die Helligkeitstransformation folgende Form:

$$\hat{r} = \Lambda(\hat{d}) = \begin{pmatrix} 0,5 & 0 & \cdots & & 0 \\ 0 & 0 & & & \\ & & s & & \vdots \\ & & & 0 & \\ \vdots & & & & s \\ & & & & & \ddots & 0 \\ 0 & & & & \cdots & 0 & s \end{pmatrix} \hat{d} + \begin{pmatrix} o_0 \\ o_1 \\ 0 \\ o_3 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{1.33}$$

Für die Abbildung dieses Beispiels müssen die Koeffizienten s, o_0, o_1, o_3 gespeichert werden. Außerdem muß dem Dekodierer übermittelt werden, welche Koeffizienten individuell kodiert worden sind.

Um zu entscheiden, welcher Spektralkoeffizient zusätzlich individuell kodiert werden soll, wird für jeden Koeffizienten die Reduktion des Approximationsfehlers bestimmt und es wird dann der verwendet, für den diese Reduktion maximal ist. Der Approximationsfehler e wird berechnet als:

$$e^2 = \|\hat{r}' - \hat{r}\| = \|\Lambda(\hat{d})\|, \tag{1.34}$$

wobei $\|\cdot\|$ eine geeignet gewählte Norm ist.

Tabelle 1.1 zeigt einen Vergleich von konventioneller Fraktalkodierung, FTC-Kodierung und DCT-Kodierung.

1.9.2 Kodierung

Um ein Bild zu kodieren, wird in quadratische Range-Blocks eingeteilt. Für jeden Range-Block wird der Domain-Block gesucht, so daß die Spektralkoeffizienten des Range-Blocks möglichst gut über eine Helligkeitstransformation der Form (1.32)

FBC Fraktale Block-Kodierung	**FTC** Fraktale Transformations-Kodierung	**TC** Transformations- Kodierung
$\hat{r} = \begin{pmatrix} s\hat{d}_0 \\ s\hat{d}_1 \\ s\hat{d}_2 \\ \vdots \\ s\hat{d}_l \\ \vdots \\ s\hat{d}_{N^2-1} \end{pmatrix} + \begin{pmatrix} o \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix}$	$\hat{r} = \begin{pmatrix} 0,5 \cdot \hat{d}_0 \\ 0 \\ s\hat{d}_2 \\ 0 \\ s\hat{d}_l \\ \vdots \\ s\hat{d}_{N^2-1} \end{pmatrix} + \begin{pmatrix} o_0 \\ o_1 \\ \vdots \\ o_l \\ \vdots \\ o_{N^2-1} \end{pmatrix}$	$\hat{r} = \begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ \vdots \\ o_l \\ \vdots \\ o_{N^2-1} \end{pmatrix}$

Tabelle 1.1: Vergleich der Approximationsschemata von konventioneller Fraktalkodierung (Fractal Block Coding - FBC), Fraktaler Transformationskodierung (Fractal Transform Coding - FTC) und Transformationskodierung (Tranform Coding - TC; zum Beispiel DCT-Kodierung)

wiedergegeben werden. Um ein gutes Verhältnis von Kompression zu Bildqualität zu erreichen, sollte die Komplexität der Transformation dem Bildinhalt angepaßt sein.

Dazu wird zuerst eine Helligkeitstransformation erster Ordnung verwendet (kein Spektralkoeffizient wird individuell kodiert). Durch eine solche Transformation können in der Regel einfach strukturierte Blöcke approximiert werden. Der Domain-pool wird nach dem letzten in Kapitel 1.4 beschriebenen Schema ausgewählt. Zunächst wird nur ein kleiner Domain-pool betrachtet. Wenn der Approximationsfehler zu groß ist, muß entweder der Domain-pool vergrößert werden, oder eine Helligkeitstransformation höherer Ordnung verwendet werden.

Durch die variable Komplexität des Verfahrens muß natürlich mehr Information gespeichert werden, aber die erhöhte Bildqualität kann diesen Mehraufwand an Speicherplatz wohl kompensieren. Der Rechenaufwand, um die optimale Kodierung nach diesem Schema zu bestimmen, ist sehr hoch. In [12] wird eine suboptimale Methode zur Bestimmung der Kodierung angegeben.

Es wird angefangen mit einem Domain-pool, der nur den Domain-Block über dem zu kodierenden Range-Block enthält (Kapitel 1.4 letzter Teil, Punkt 1), und mit einer Helligkeitstransformation der Ordnung 1 (1.31). Wenn der Approximationsfehler zu groß ist, wird schrittweise die Komplexität der Kodierung erhöht.

Es wird nun folgendermaßen vorgegangen. Sei zunächst B_i die Anzahl der Bits, die nötig ist um R_i mit einer bestimmten Transformation zu kodieren, und F_i der Approximationsfehler des Blocks R_i. Zunächst wird für alle R_i die optimale Helligkeitstransformation erster Ordnung mit einem Domain-pool der Suchtiefe 0 bestimmt. Für alle diese R_i wird B_i und F_i bestimmt. In jedem weiteren Schritt wird der Code für genau einen Block geändert. Es wird immer der Block ausgewählt, für den der Approximationsfehler bei dem aktuellen Stand der Kodierung am größten ist. Es kann nun entweder der Domain-pool für diesen Range-Block vergrößert werden oder die Ordnung der Helligkeitstransformation um 1 erhöht werden. Es wird die Änderung der Kodierung asugewählt, für die

$$\frac{F(c_{alt}) - F(c_{neu})}{B(c_{neu}) - B(c_{alt})} \tag{1.35}$$

maximal ist, wobei c_x den besten Code der jeweiligen Komplexität für den Block R_i bezeichnet.

Diese Prozedur wird solange fortgesetzt bis eine gewünschte Kompressionsrate erreicht ist oder ein bestimmter Approximationsfehler unterschritten wird.

Wenn die Ordnung der Transformation erhöht wird, wird neu nach dem besten Domain-Block aus den Suchregionen 1 und 2 gesucht (siehe Kapitel 1.4). Es wird außerdem der Koeffizient individuell kodiert, der für den größten Fehler bei der letzten Approximation verantwortlich war.

Die Transformationsparameter werden zur Speicherung entropiekodiert.

1.9.3 Ergebnisse

Es wurden Experimente mit dem Testbild Lenna in der Größe 512×512 Pixel gemacht. Dabei hat sich ergeben, daß die FTC-Kodierung der JPEG-Kodierung und einem Fraktalkodierer mit HV-Unterteilung bei allen Kompressionsraten überlegen ist. Über die Kompressionszeit wird allerdings nichts gesagt, nur daß die Komplexität des Verfahrens sehr hoch ist.

1.10 Vergleich der verschiedenen Bildkompressionstechniken

1.10.1 Vergleiche in der Literatur

In [19] findet sich ein Vergleich zwischen dem JPEG-Algorithmus und einem fraktalen Kodierer von Iterated Systems. Cheung et al. haben herausgefunden, daß JPEG bei allen Kompressionsraten besser als das Programm von Iterated Systems ist. Dieser Vergleich wurde allerdings 1991 durchgeführt, also zu einem Zeitpunkt, an dem die Fraktalkompression noch nicht sehr weit entwickelt war.

Y. Fisher et al. haben 1994 einen Vergleich von fraktalen Methoden mit JPEG und Wavelet(epic) durchgeführt [28]. Dabei haben sie einen Quadtree-Kodiererer und einen HV-Kodierer verwendet. Als Testbild haben sie "Lenna" in der Größe 512×512 Pixel verwendet. Es hat sich ergeben, daß bei niedriger Bildqualität Wavelet-Kompression den höchsten PSNR-Wert ergibt, bei mittlerer Kompression sind die PSNR-Werte von HV-Partition, JPEG- und Wavelet-Kompression etwa gleich. Bei hoher Kompression (Faktor 30 bis über 100) ist die Bildqualität beim HV-Kodierer am größten. Der Quadtree-Kodierer liefert fast bei allen Kompressionsraten die niedrigste Bildqualität und bei keiner Rate die beste Bildqualität. Die Kompressionszeiten sind in dieser Untersuchung allerdings nicht in Betracht gezogen worden.

G.J. Ewing et al. haben untersucht, wie gut menschliche Beobachter Objekte in Bildern erkennen können, die mit JPEG und mit einem fraktalen Kompressionsalgorithmus bei verschiedenen Qualitätsstufen komprimiert worden sind [23]. Es wird allerdings nicht gesagt, welcher Algorithmus genau verwendet wird. Sie fanden heraus, daß in mit JPEG komprimierten Bildern Objekte generell besser erkannt werden. Da allerdings nichts über den verwendeten Algorithmus gesagt wird, ist das Ergebnis nicht sehr aussagekräftig.

In [32] findet sich ein Vergleich von sehr vielen Fraktal- und Nicht-Fraktal-Kompressionstechniken. Das Testbild "Lenna" im Format 512×512 Pixel wurde verwendet. Es hat sich folgende Rangfolge der einzelnen Techniken ergeben (es werden hier nur die ersten 7 dieser Rangfolge angegeben):

1. Optimierte Wavelet-Zero-Tree-Kodierung nach Xiong et al. [65].
2. Kombinierte Fraktal-Wavelet-Kodierung [54].
3. Kombinierte Fraktal-DCT-Kodierung nach Barthel et al. (Kapitel 1.9).

4. Wavelet-Zero-Tree-Kodierung nach Shapiro [61].

5. Fraktale Kodierung mit HV-Partition (Kapitel 1.3.2).

6. DCT-Kodierung (JPEG).

7. Fraktale Kodierung mit Quadtree-Partition (Kapitel 1.3.1)

Die kombinierte Fraktal-DCT-Kodierung und Fraktal-Wavelet-Kodierung liegen dabei fast gleichauf. Außerdem sind die PSNR-Werte von HV-Kodierung und DCT-Kodierung bei niedrigen Kompressionsfaktoren fast gleich. Erst bei höheren Kompressionsraten ist die HV-Kodierung deutlich besser.

1.10.2 Eigene Untersuchungen

Für vergleichende Untersuchungen standen ein HV-Kodierer, der in [7] beschriebene geschwindigkeitsoptimierte Quadtree-Kodierer, das Programm Fractal Imager von Iterated Systems und ein JPEG-Kodierer zur Verfügung. Als Testbild wurde "Lenna" in den Auflösungen 512×512 Pixel und 256×256 Pixel verwendet.

Verständlicherweise gibt Iterated Systems nicht bekannt, was für ein Algorihmus genau in ihrem Programm verwendet wird. Das einzige, was bekannt gemacht wird, ist die Tatsache, daß ein Fraktalkodierer verwendet wird. Es ist deshalb schwierig, das Programm einzuordnen.

Abbildungen 1.13 - 1.20 zeigen das Testbild "Lenna" im Format 512×512 Pixel komprimiert mit den 4 Verfahren und jeweils 2 Kompressionsraten. Man kann gut erkennen, daß die Fraktalkompression mit Quadtree-Partition den anderen Verfahren deutlich unterlegen ist. Bei niedriger Kompression ist wenig Unterschied zwischen HV-Partition, JPEG und Fractal Imager zu erkennen, wobei Fractal Imager vielleicht doch noch einen Tick besser als HV und JPEG ist. Bei hoher Kompression ist JPEG den fraktalen Verfahren deutlich unterlegen. Die visuelle Bildqualität, die die HV-Partition liefert, ist bei hoher Kompression Fractal Imager etwas unterlegen.

Die Kompressionszeiten lagen mit Fractal Imager auf einem Pentium-133 bei etwa 30 Sekunden, mit dem Quadtree-Kodierer von B. Bani-Eqbal [7] bei etwa 17 Sekunden auf einer SUN SPARCstation-10, und mit dem HV-Kodierer (optimiert durch Approximate-Nearest-Neighbor-Searching, siehe Kapitel 1.6.2 und einer Domain-pool-Gitterweite 6, siehe Kapitel 1.4) bei etwa 7 Minuten auf Pentium-133 (Durch verschiedene Einstellungen im Programm kann mit dem HV-Kodierer Kompressionszeiten zwischen etwa 30 Sekunden und 20 Minuten erreichen. Bei längerer Kompressionszeit ist die Bildqualität geringfügig besser) . Die Kompression mit dem JPEG-Algorithmus dauert etwa 4 Sekunden auf Pentium-133. Die Dekompression dauert bei allen Verfahren wenige Sekunden, nur bei dem Quadtree-Kodierer dauert sie etwa 3-4 mal so lang, da der Dekodieralgorithmus in B. Bani-Eqbals Code nicht optimiert ist.

Es hat sich also gezeigt, daß die Fraktale Bildkompression mit HV-Partition und Fractal Imager bei niedrigen Kompressionsraten in Bezug auf Bildqualität geringfügig besser und bei hohen Kompressionsraten (über 50) deutlich besser als JPEG sind. Es werden außerdem höhere Maximalraten erreicht (man könnte im Prinzip ein Bild auch durch 4 kontrahierende Abbildungen kodieren, der Approximationsfehler wäre allerdings dann sehr hoch). Die Kompressionszeiten sind deutlich höher wie mit JPEG (ein paar Minuten gegenüber wenigen Minuten), diese läßt sich aber in gewissen Maßen noch kontrollieren mit Hilfe der Gitterweite l (siehe Kapitel 1.4).

Abbildung 1.13: “Lenna”, fraktal komprimiert mit HV-Partition: 16250 Bytes, Kompressionsfaktor 16,1, PSNR-Wert 34,22 dB

Abbildung 1.14: “Lenna”, fraktal komprimiert mit Quadtree-Partition: 16121 Bytes, Kompressionsfaktor 16,2, PSNR-Wert 33,01 dB

Abbildung 1.15: “Lenna”, fraktal komprimiert mit Fractal Imager: 15710 Bytes, Kompressionsfaktor 16,7, PSNR-Wert: 34,91 dB

Abbildung 1.16: “Lenna”, mit JPEG komprimiert, 16563 Bytes, Kompressionsfaktor 15,8, PSNR-Wert 35,98 dB

Abbildung 1.17: “Lenna”, fraktal komprimiert mit HV-Partition: 5367 Bytes, Kompressionsfaktor 48,8, PSNR-Wert 29,75 dB

Abbildung 1.18: “Lenna”, fraktal komprimiert mit Quadtree-Partition: 5395 Bytes, Kompressionsfaktor 48,6, PSNR-Wert 27,29 dB

Abbildung 1.19: “Lenna”, fraktal komprimiert mit Fractal Imager 5242 Bytes, Kompressionsfaktor 50,0, PSNR-Wert 30,59 dB

Abbildung 1.20: “Lenna” mit JPEG komprimiert: 5323 Bytes, Kompressionsfaktor 49,2, PSNR-Wert 27,94 dB

Kapitel 2

Fraktale Videokodierung

Die Fraktale Bildkompression kann auf die Kompression von Bildsequenzen erweitert werden, allerdings ist die Fraktale Videokodierung noch nicht sehr weit entwickelt. Es wurden verschiedene Vorschläge gemacht [1, 2, 3, 5, 13, 14, 15, 24, 30, 31, 34, 36, 39, 37, 45, 46, 50, 52, 53, 63].

1. Um ein Video zu kodieren, wird nur das erste Bild der Sequenz fraktal kodiert. Die weiteren Bilder werden durch Abbildungen von einem Bild (Frame) zum nächsten Bild der Sequenz kodiert. Dadurch, daß die Unterschiede von einem Frame zum nächsten Frame in Videosequenzen meistens gering sind, kann man ein Frame in der Regel durch wenig Abbildungen kodieren, was eine hohe Kompression ergibt. Weiterhin können nun auch nichtkontrahierende Abbildungen verwendet werden, da nicht mehr iteriert werden muß. Außerdem hat diese Art der Kodierung den Vorteil, daß die Dekodierung nur noch einen Schritt für jedes Frame braucht. Diese Methode ist eine Inter-Frame-Kodierung

   In einer Variation dieses Verfahrens betrachtet man zusätzlich Abbildungen von Domains desselben frames auf den jeweiligen Range-Block. Dies ist eine Kombination aus Intra- und Inter-Frame-Kodierung.

2. Die Videodaten werden als 3-D-Information aufgefaßt (die 3. Dimension ist die Zeit). Zur fraktalen Kodierung eines Videos werden affine Abbildungen in 3 Dimensionen verwendet. Man teilt das Video in verschiedene Range-Blocks auf, die diesmal natürlich 3-dimensional sind, also zum Beispiel Würfel. Nun sucht man aus einem Domain-pool von 3-D-Blöcken dann den Block, der über eine affine Helligkeitstransformation die Pixel des Range-Blocks wiedergibt. Die gefundenen Transformationen kodieren das Video.

Es ist natürlich auch möglich, jeden Frame einzeln fraktal zu kodieren. Dadurch nutzt man allerdings nicht die Ähnlichkeit zwischen den einzelnen Bildern aus. Die erreichbaren Kompressionsraten werden also wesentlich geringer sein.

2.1 3-D-Videokodierung

Zu dieser Methode, Videos fraktal zu kodieren, standen leider nur 3 Literaturquellen zur Verfügung [13, 36, 39]. Das Grundprinzip ist bei allen 3 Verfahren, Abbildungen von 3-dimensionalen Domain-Blocks auf 3-dimensionale Range-Blocks zu bestimmen (siehe Abb. 2.1). Es folgt eine Zusammenfassung der einzelnen Artikel.

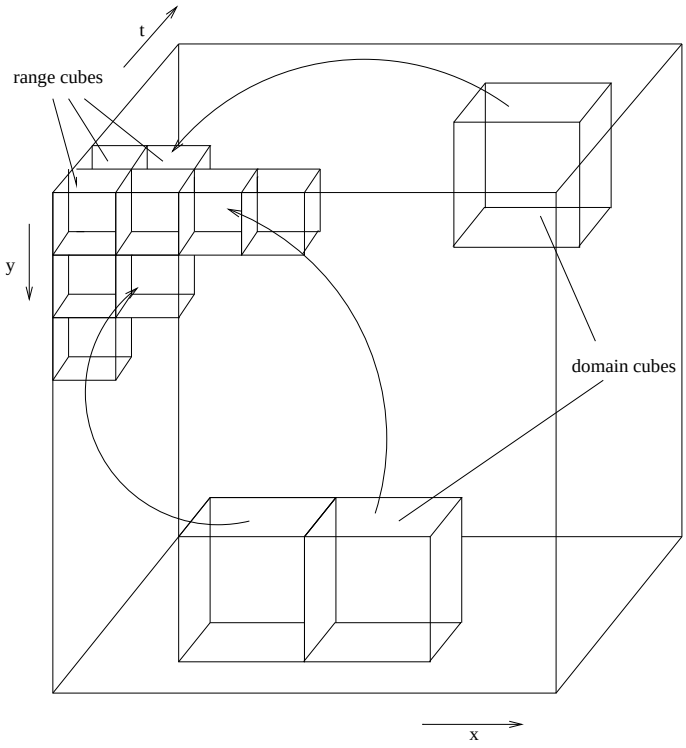

Abbildung 2.1: Prinzip der 3-D-Video-Kodierung

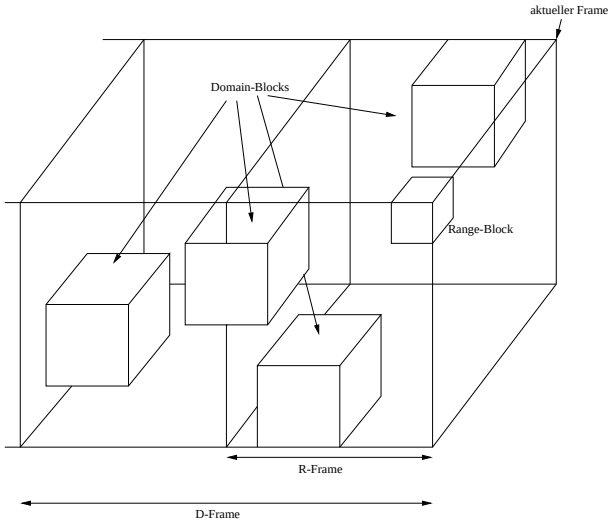

Abbildung 2.2: Einteilung einer Bildsequenz in R-Frames und D-Frames

2.1.1 Verfahren von Lazar und Bruton

In [36] beschreiben Lazar und Bruton einen Algorithmus, der 3-dimensionale Transformationen zur Kodierung eines Videos verwendet. Ein 3-dimensionaler Range-Block wird über eine affine kontrahierende Transformation durch einen Domain-Block approximiert.

Zunächst gibt sich bei der Kodierung eines Videos das Problem, daß die Größe in der Zeitdimension im Gegensatz zur Größe in den räumlichen Dimensionen nahezu unbeschränkt ist. Wenn man nun versucht, das gesamte Video in Range-Blocks und Domain-Blocks einzuteilen, wird man erstens massive Probleme mit der Rechenzeit und außerdem auch noch Speicherplatzprobleme bekommen, da das ganze Video auf einmal im Speicher gehalten werden muß . Dieselben Probleme hat man dann auch bei der Dekodierung. Ein Video könnte erst dann dekodiert werden, wenn sämtliche Daten übertragen worden sind. Lazar und Bruton haben zur Lösung dieses Problem den folgenden Vorschlag gemacht.

Die maximale Größe der Range-Blocks sei $B \times B \times T$, wobei B die maximale räumliche Länge angibt, und T die maximale zeitliche. Die maximale Größe der Domain-Blocks sei $2B \times 2B \times 2T$. Das Video wird zunächst in sogenannte R-Frames eingeteilt. Dieses sind zeitliche Abschnitte, die aus kT Einzelbildern bestehen, wobei k eine ganze Zahl ist. Es passen also eine ganze Anzahl von Range-Blocks in den R-Frame. Die Domain-Blocks werden aus dem sogenannten D-Frame ausgewählt. Dieser kann vor dem R-Frame anfangen, muß aber an derselben zeitlichen Stelle, wie der R-Frame enden. Die Länge eines D-Frames ist $2lT$, wobei l eine ganze Zahl ist. Abbildung 2.2 zeigt eine solche Unterteilung in R-Frames und D-Frames. Man beachte, daß Domain-Blocks im Falle $l > k$ auch außerhalb des R-Frames liegen können (in Abb. 2.2 der am weitesten links liegende Beispiel-Domain-Block). Zur Dekodierung muß in diesem Fall nur einmal die Transformation angewendet werden, da die Pixel im Domain-Block schon vollständig dekodiert wurden.

Zur Partition eines R-Frames in Range-Blocks haben Lazar und Bruton folgendes Verfahren, das eine Variante des Quadtreeverfahrens ist, angewendet. Ein R-Frame wird zunächst in Quader der Größe $B \times B \times T$ eingeteilt. (Die räumlichen Ausdehnungen des Videos sollten natürliche Vielfache von B sein.) Nun wird für jeden Range-Block R geprüft, ob ein geeigneter Domain-Block D aus dem Domain-pool

existiert, so daß der Approximationsfehler T nach (1.14) unter einem Grenzwert T_{max} liegt. Falls dies der Fall ist, wird nicht weiter unterteilt, und das gefundene D zusammen mit den Parameter s und o für die Kodierung verwendet. Falls der Fehler größer ist, wird entweder räumlich in 4 kleinere Range-Blocks aufgeteilt oder zeitlich in 2 kleinere Range-Blocks. Zeitlich wird geteilt, wenn die Verteilung des Approximationsfehlers nicht gleichmässig über die Einzelbilder des Range-Blocks verteilt sind, und räumlich wird im anderen Fall geteilt.Die Differenz zwischen dem maximalen und minimalen Approximationsfehler eines Einzelbildes dient dabei als Indikator für die Gleichmäßigkeit dieser Verteilung. Für jeden Frame des Range-Blocks wird der mittlere quadratische Fehler des kodierten zum Original-Frame berechnet. Wenn der Unterschied zwischen dem maximalen und minimalen Wert unter einer Grenze liegt, wird räumlich geteilt, sonst zeitlich. Außerdem wird eine maximale Anzahl von räumlichen Teilungen (N_s) und zeitlichen Teilungen (N_t) festgelegt.

Für 3-D-Blöcke sind wesentlich mehr Isometrien möglich, D auf R abzubilden, als bei 2-D-Blöcken. Die Isometrie kann für die 3-D-Blöcke aufgeteilt werden in eine Intra-Frame-Isometrie und eine Inter-Frame-Isometrie. Für die Intra-Frame-Isometrie werden die 8 Isometrien zugelassen, ein Quadrat auf ein Quadrat abzubilden. Für die Inter-Frame-Isometrie gibt es 2 Möglichkeiten: die Identität und die Ivertierung der zeitlichen Abfolge. Es werden nicht alle Möglichkeiten zugelassen, diese Isometrien zu kombinieren, sondern es wird entweder eine Intraframe oder eine Interframe-Isometrie angewendet. Zur Kodierung sind dann 3 Bits bzw. 1 Bit für die Intra-Frame- bzw. Inter-Frame-Isometrie nötig.

Da die Kompressionszeit sehr stark von der Größe des Domain-pools abhängt, ist es nicht sinnvoll, alle möglichen Domain-Blocks in die Suche einzubeziehen. Im 3-D-Fall wird der Domain-pool dadurch noch wesentlich größer als im 2-D-Fall. Man muß die Größe des Domain-pools also möglichst sinnvoll einschränken. Lazar und Bruton betrachten hierfür nur die Domain-Blocks in der Nähe des zu kodierenden Range-Blocks. Die andere Möglichkeit wäre, die Gitterweite l des Gitters, auf dem die Domain-Blocks liegen, sehr groß zu wählen (siehe auch Kapitel 1.4).

Lazar und Bruton haben den beschriebenen Algorithmus mit verschiedenen Parametern mit den Sequenzen "Miss America" und "Travelling Salesman" getestet. Bei einem durchschnittlichen Kompressionsfaktor von 74,39 ("Miss America") bzw. 41,8 ("Travelling Salesman") haben sich PSNR-Werte von 32-34 dB bzw. ungefähr 28 dB ergeben. Die Kompressionszeiten lagen allerdings relativ hoch. Lazar und Bruton geben diese mit der Anzahl der nötigen Gleitkommaoperationen an. Für "Miss America" waren dies $2,9 \times 10^9$ Additionen und $1,3 \times 10^9$ Multiplikationen.

2.1.2 3-D-Videokodierung mit der Bath Fractal Transform

In [39] wird eine Beschreibung eines Videokodierers gegeben, der sich auf die Bath Fractal Transform, abgek. BFT (siehe Kapitel 1.8) stützt. Im Gegensatz zu Monro et al., die in ihrem Videokodierer jedes Bild einzeln mit der BFT zu kodieren (siehe auch Kapitel 2.3) wenden Li, Novak und Forchheimer eine 3-D-Version der BFT an.

Das Video wird zunächst in 3-D-Blöcke aufgeteilt, zum Beispiel der Größe $8 \times 8 \times 8$ Pixel. Diese Blöcke sind die Domain-Blocks D. Diese werden in 8 kleinere Blöcke R_i aufgeteilt. Es werden dann kontrahierende Transformationen des größeren Blocks D auf jedes R_i gesucht, so daß die R_i möglichst gut approximiert werden. Eine solche Transformation ω besteht aus einem Operator G, der den Block D auf die Größe von R verkleinert, einer Isometrie-Operation S und einer Helligkeitstransformation H wie bei der 2-dimensionalen BFT auch.

$$\omega = H \circ S \circ G \tag{2.1}$$

Sei im folgenden $I(x, y, t)$ die Helligkeit eines Pixel am geometrischen Ort (x, y) zum Zeitpunkt t relativ zum Beginn des durch $S \circ G$ transformierten Domain-Blocks. $J(x, y, t)$ sei der Pixelwert an der Stelle (x, y, t) des durch H approximierten Range-Blocks.

$$J(x, y, t) = H(x, y, t, I(x, y, t)) \tag{2.2}$$

Für H werden Polynome verwendet. Die konventionelle Fraktalkodierung verwendet Polynome der Ordnung 0:

$$H(x, y, t, I(x, y, t)) = p_0 I(x, y, t) + p_1 \tag{2.3}$$

Der Parameter p_0 entspricht dem Parameter s der konventionellen Fraktalkodierung und p_1 entspricht o.

Wenn ein Polynom erster Ordnung verwendet wird, hat H die Form:

$$H(x, y, t, I(x, y, t)) = p_0 I(x, y, t) + p_1 + p_2 x + p_3 y + p_4 t \tag{2.4}$$

Und für ein Polynom zweiter Ordnung ergibt sich:

$$\begin{aligned} H(x, y, t, I(x, y, t)) &= p_0 I(x, y, t) + p_1 + p_2 x + p_3 y + p_4 t \\ &+ p_5 x^2 + p_6 y^2 + p_7 t^2 + p_8 xy + p_9 xt + p_{10} yt \end{aligned} \tag{2.5}$$

Bei der Kodierung sind die optimalen Parameter $p_0, \ldots, p_m$ zu bestimmen. Im Falle Polynome erster Ordnung muß

$$\begin{pmatrix} I(x_1, y_1, t_1) & 1 & x_1 & y_1 & t_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ I(x_n, y_n, t_n) & 1 & x_n & y_n & t_n \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} J(x_1, y_1, t_1) \\ \vdots \\ J(x_n, y_n, t_n) \end{pmatrix} \tag{2.6}$$

nach $p_0, \ldots, p_4$ gelöst werden, wobei n die Anzahl der Pixel im Range-Block ist. Gleichung (2.6) kann auch geschrieben werden als:

$$Rp = c\,, \tag{2.7}$$

wobei $c = (J(x_1, y_1, z_1), \ldots, J(x_n, y_n, z_n))$, $p = (p_0, \ldots, p_4)$ und R die Matrix in (2.6) ist. Da die Approximation in der Regel nicht exakt ist, hat Gleichung (2.6) im allgemeinen keine Lösung. Um den Vektor p zu bestimmen, so daß $||Rp - c||$ minimal ist, muß

$$R^T Rp = R^T c\,, \tag{2.8}$$

nach p gelöst werden (Die Lösung dieses Gleichungssystems existiert und ist eindeutig bestimmt). p ergibt sich als

$$p = (R^T R)^{-1} R^T c \tag{2.9}$$

Li et al. haben Experimente mit zwei verschiedenen Unterteilungen des Videos gemacht. Als Testvideo wurden die ersten 16 Frames von "Miss America" verwendet.

Allerdings wurden noch keine Untersuchung zur Quantisierung und Entropiekodierung der BFT-Parameter gemacht. Deshalb konnte auch nur ein ungefährer Kompressionsfaktor angegeben werden, der das Verhältnis der Anzahl der BFT-Parameter zur Anzahl der Pixel beschreibt.

Unterteilung I Das Video wird $8 \times 8 \times 8$ Domain-Blocks aufgeteilt. Diese 3-D-Blocks sind die Domain-Blocks. Jeder dieser 3-D-Blöcke wird weiter in 8 Range-Blocks der Größe $4 \times 4 \times 4$ Pixel aufgeteilt. Für H werden Polynome erster Ordnung verwendet. Nach Gleichung (2.9) werden die Parameter der Transformation des Domain-Block auf den jeweiligen Range-Block bestimmt. Li et al. haben nur eine einzige Isometrie - die Identität - verwendet, was nach den Ergebnissen von Monro et al. auch sinnvoll ist (siehe Kapitel 1.8). Es müssen also nur die Parameter $p_0, \ldots, p_4$ gespeichert werden. Der mittlere SNR-Wert der dekodierten Sequenz war 37.63 dB und der ungefähre Kompressionsfaktor 13.

Unterteilung II Um die Kompressionsrate weiter zu erhöhen, haben Li et al. größere Blöcke verwendet. Das Video wird in $16 \times 16 \times 8$ Domain-Blocks aufgeteilt und jedes dieser Domain-Blocks wiederum in 8 Range-Blocks der Größe $8 \times 8 \times 4$ Pixel. Für H wird wieder eine Polynomfunktion erster Ordnung verwendet. Der mittlere SNR-Wert der Testsequenz war 27.21 dB, der ungefähre Kompressionsfaktor 51. Li et al. berichten, daß die Sequenz sehr blockhaft wirkt und geben zu bedenken, daß die Polynomfunktion erster Ordnung wohl nicht ausreichend ist, um die groß en Blocks zu approximieren. Es wurden deshalb auch Polynome zweiter Ordnung verwendet. Der mittlere SNR-Wert der dekodierten Testsequenz war nun 36.22 dB und der ungefähre Kompressionsfaktor 25. Verglichen mit der ersten Unterteilung hat sich also durch letzteres Verfahren die Bildqualität nur um etwa 1dB erniedrigt und die Kompression fast verdoppelt.

Untersuchung einer Sequenz mit starker Bewegung Das Experiment wurde bei einer Teilsequenz von "Miss America" mit starker Bewegung (Frames 36-43) wiederholt mit dem Polynom zweiter Ordnung. Li et al. haben festgestellt, daß hier die Bildqualität geringer wird. Es wird vermutet, daß die BFT nicht so gut in der Lage ist, größere Bewegung genau genug zu kodieren. Es ist deshalb ein 3-D-BFT-Kodierer mit Motion Compensation vorgeschlagen worden [39]. Dieser ist allerdings nicht implementiert worden.

2.1.3 3-D-FTC-Kodierung

Barthel et al. haben die von ihnen entwickelte FTC-Kodierung (siehe Kapitel 1.9) auf die Kodierung von Videosequenzen erweitert durch Einführung einer dritten Dimension [13]. Außerdem haben sie auch Experimente gemacht mit einer kombinierten Wavelet-Fraktal-Kompression [14]. Bei letzterem Ansatz wurde die DCT-Transformation der Range-Blocks und Domain-Blocks durch eine Wavelet-Transformation ersetzt. Durch die Fraktale Transformation werden Wavelet- Koeffizienten des Domain-Blocks auf Wavelet-Koeffizienten des Range-Blocks abgebildet.

Kombination mit DCT-Kodierung

Der von Barthel et al. [13] vorgestellte Ansatz weicht eigentlich nur darin von der 2-D-FTC-Kodierung ab, daß auf die Range-Blocks (Ausmaße $N \times N \times N$ Pixel) und Domain-Blocks die 3-dimensionale DCT angewendet wird. Die Spektralkoeffizienten werden wieder Zick-Zack gescannt und in einem Vektor der Dimension N^3 zusammengefaßt. Die Domain-Blocks werden aus einem kleinen Bereich um den Range-Block gesucht, der eventuell vergrößert werden kann. Der restliche Kodieralgorithmus funktioniert genauso wie die in Kapitel 1.9 beschriebene 2-D-FTC-Kodierung.

Barthel et al. haben Experimente mit einem ersten sehr einfachen Kodierer gemacht. Es wurden range cubes der Größe $4 \times 4 \times 4$ Pixel verwendet und Domain-pool

der Größe 256. Bei einer Bitrate von 80 kBit/s hatte die QCIF-Sequenz (Bildrate 25 Hz) "akzeptable Qualität". Ein SNR-Wert wird nicht angegeben.

Kombination mit Wavelet-Kodierung

In [14] wird ein kombinierter Wavelet/Fraktal-Kodierer vorgestellt. Leider beschreiben Barthel et al. den Algorithmus nicht sehr ausführlich.

Die Verwendung der Wavelet-Transformation statt der DCT in einem kombinierten Fraktal-Transformationskodierer bietet den Vorteil, daß die Wavelet-Transformation keine Blockartifakte erzeugt wie die DCT-Transformation.

Die Diskrete Wavelet-Transformation eines Bildes erzeugt eine Repräsentation, die das Bild in verschiedenen Maßstäben darstellt [41]. Die Wavelet-Transformation kann dabei als eine mehrstufige Pyramide angesehen werden, wobei die Koeffizienten, die den gröbsten Maßstab repräsentieren, oben in der Pyramide und die Koeffizienten für den feinsten Maßstab unten stehen.

Das Prinzip des Algorithmus ist dasselbe wie bei der kombinierten DCT-Fraktalkodierung. Es werden die 3-D-Wavelet-Koeffizienten der Range-Blocks R_i und der Domain-Blocks D_i berechnet und anschließend kontrahierende Abbildungen gesucht, durch die die Koeffizienten von R_i möglichst gut durch die Koeffizienten von D_i approximiert werden können. Wenn die Approximation nicht genau genug ist, werden manche Koeffizienten nach dem in Kapitel 1.9 beschriebenen Verfahren einzeln kodiert. Barthel et al. haben für Standbildkompression die besten Resultate mit QMF-Wavelets von Johnston [35] erhalten.

Um die Kontraktivität der Abbildungen zu erhalten, werden Wavelet-Koeffizienten eines bestimmten Maßstabes durch Koeffizienten des nächstgröberen Maßstabes approximiert. Zur Quantisierung der einzelnen Koeffizienten wurden keine Angaben gemacht.

Barthel et al. haben einen Kodierer implementiert und berichten, daß die erhaltene Bildqualtät selbst bei hohen Kompressionsraten gut ist. Sie präsentieren aber keine konkreten Kompressionsraten und PSNR-Werte. Das Verfahren eignet sich wegen der hohen Rechenzeit nach Barthel et al. nur für Videospeicherung, aber nicht für Echtzeitanwendungen.

2.2 Fraktale Inter-/Intra-Frame-Kodierung

Der Ansatz, Videos durch Abbildungen von einzelnen Frames in sich, oder durch Abbildungen von einem Frame zum nächsten zu kodieren, wurde ist schon ein bißchen weiter entwickelt wie die 3-D-Videokodierung [3, 5, 24, 30, 31, 34, 37, 52, 53] zur Verfügung. Der Decoder zu der in [24] beschriebenen Implementation (lauffähig auf SGI) ließ sich mit 2 Beispielsequenzen aus dem Internet herunterladen (http://inls3.ucsd.edu/y/Fractals/Video/fracvideo.html). In [28] und [5] sind reine Inter-Frame-Kodierer beschrieben, sonst handelt es sich um kombinierte Inter-/Intra-Frame-Kodierer oder Variationen bzw. Weiterentwicklungen davon.

2.2.1 Reine Inter-Frame-Kodierer

Da die Beschreibung in [5] sehr spärlich ist, sind die folgenden Ausführungen auf [24] gestützt. Es sei das Einzelbild f_i und das vorhergehende dekodierte Einzelbild g_{i-1} der Sequenz gegeben. Es werden nun Transformationen von g_{i-1} auf f_i gesucht. Dadurch werden keine Selbstähnlichkeiten eines Einzelbildes mehr kodiert, sondern Ähnlichkeiten zwischen den Bildern. Der Kodierer ist aber immer noch fraktal, in dem Sinne, daß die meisten Abbildungen (nicht unbedingt alle) kontrahierend sind und das Video auflösungsunabhängig kodiert wird.

Eine Quadtree-Unterteilung wird auf f_i angewendet. Um einen Range-Block R_j zu kodieren, gibt es 2 Möglichkeiten:

1. Die Pixel von R_j werden einfach aus einem gleichgroßen Block von g_{i-1} kopiert. Typischerweise wird der Block verwendet, der an derselben räumlichen Position wie R_j oder nicht weit davon entfernt liegt.

2. R_j wird durch einen Domain-Block aus g_{i-1} kodiert, dessen Seitenlängen doppelt so lang wie die von R_j sind. Zur Approximation wird eine Transformation der Form 1.6 verwendet. Der Domain Block wird aus einem Domain-pool genommen, der gleichmäßig über g_{i-1} verteilt ist (Gitterweite l, siehe auch Kapitel 1.4. Die Auswahl erfolgt nach dem in Kapitel 1.2.1 beschriebenen Verfahren.

Es wird zunächst versucht, die erste Möglichkeit anzuwenden, da dabei weniger Bits benötigt werden. Wenn der Approximationsfehler größer als ein bestimmter Maximalwert t_{mc} ist, wird versucht die zweite Möglichkeit anzuwenden. Wenn nun der Approximationsfehler wieder über einem Maximalwert t_e liegt, wird nach der Quadtree-Methode eine Stufe weiter unterteilt und es wird für jedes der neuen R_i versucht wieder die erste Methode anzuwenden. Es wird maximal bis zu einer bestimmten Minimalgröße der R_j unterteilt.

Zur Beschleunigung der Suche des optimalen Domain-Blocks verwenden Fisher et al. das in Kapitel 1.6.1 beschriebene Verfahren.

Bei diesem Kodierungsschema bleibt noch das Problem übrig, wie das erste Bild der Sequenz kodiert werden soll, da für f_1 g_0 nicht existiert. Hierzu machen Fisher et al. 2 Vorschläge:

1. f_1 wird mit einem herkömmlichen fraktalen Schema kodiert. Dieser Vorschlag wird auch in [5] gemacht.

2. Einfach das Problem ignorieren und $g_0 = 0$ setzen. Dadurch ist der Approximationsfehler des ersten Bildes g_1 natürlich sehr groß . Er wird aber sehr schnell kleiner werden.

Es wurde der 2. Vorschlag implementiert. Es hat sich ergeben, daß schon bei f_2 keine merklichen Approximationsfehler mehr auftreten.

Fisher et al. haben den Algorithmus mit 20 verschiedene Parametereinstellungen und der Testsequenz "Miss America" ausgetestet. Bei der höchsten erreichten Bildqualität mit einem durchschnittlichen PSNR-Wert von 37,01 dB ergab sich ein Kompressionsfaktor von 24,95 und der höchste Kompressionsfaktor war 244,29 mit einem durchschnittlichen PSNR-Wert von 31,44 dB. Die Kompressionszeiten liegen zwischen etwa 2,5 sec und 66 sec auf einer Silicon Graphics Personal Iris 4D/35, wobei die unterschiedlichen Zeiten mit verschiedenen Parametereinstellungen erreicht werden (66 sec bei sehr guter Qualität). Bei der Dekodierung werden 5 bis 12 Frames pro Sekunde erreicht.

Da der Dekoder und 2 Testsequenzen zur Verfügung standen, konnte mit einem H.263 Kodierer (DCT-basiert) verglichen werden. Die mit dem H.263-Kodierer erreichte visuelle Bildqualität ist wesentlich besser als die der beiden fraktalen kodierten Testsequenzen bei etwa demselben Kompressionsfaktor. Blockeffekte sind bei dem Fraktalkodierer deutlich sichtbar. Man muß alerdings bedenken, daß der von Fisher implementierte Algorithmus ein erster Versuch war und an DCT-basierten Verfahren schon seit Jahrzehnten geforscht und entwickelt wird.

2.2.2 Kombinierte Inter-/Intra-Frame-Kodierung

In [3, 34, 37] werden einfache kombinierte Inter-/Intra-Frame-Kodierer vorgestellt. Die Funktionsweise ist bei den beschriebenen Verfahren ähnlich.

In den 3 Arbeiten wird zwischen Vordergrund und Hintergrundbild unterschieden. Das Hintergrundbild besteht aus den Bereichen, in denen es sehr wenig Bewegung gibt, während das Vordergrundbild aus den Bereichen besteht, in denen sich von Frame zu Frame relativ viel ändert. In der Sequenz eines Nachrichtensprechers wäre zum Beispiel das Vordergrundbild der Nachrichtensprecher selbst und das Hintergrundbild eben der Hintergrund.

Ein Frame wird in Blöcke (Range-Blocks) eingeteilt. Wenn diese Blöcke zum Hintergrundbild gehören, werden diese Blöcke einfach als Kopien des jeweiligen Blocks des letzten Frames an derselben räumlichen Position kodiert. Die anderen Domain-Blocks werden mit einem Quadtree-Schema fraktal kodiert, wobei der Domain-pool aus demselben Frame wie der zu kodierende Range-Block stammt. In [37] werden außerdem die Domain-Blocks nur aus dem Vordergrundbild genommen.

Als Kriterium für die Einteilung eines Blocks in Vordergrund- und Hintergrundbild findet die mittlere quadratische Differenz der Pixel dieses Blocks in aufeinanderfolgenden Frames Verwendung. Falls diese Differenz größer als ein bestimmter Maximalwert ist, gehört der Block zum Vordergrundbild, im anderen Fall zum Hintergrundbild.

In [3] wird noch eine dritte Art von Transformation, Blocks zu kodieren verwendet. Diese geometrische Transformation, die in [3] leider nicht formell angegeben wird, beschreibt Inter-Frame-Veränderungen. Vor allem Bewegungen können durch diese Transformation gut kodiert werden (Motion Compensation).

Bei der Kodierung wird zunächst geprüft, ob einfach aus dem letzten Frame kopiert werden kann, dann wird versucht, die im letzten Abschnitt beschriebene geometrische Transformation zu verwenden und wenn der Approximationsfehler immer noch über einem Maximalwert liegt, wird schließlich eine konventionelle Fraktalkodierung verwendet. Wenn alle 3 Möglichkeiten einen zu großen Approximationsfehler liefern, wird weiter nach dem Quadtree-Schema unterteilt.

In [34] wurden Experimente mit der Graustufen-Testsequenz "Miss America" im Format 352 × 280 Pixel (CIF) und 25 Hz Frame-Rate durchgeführt. Es wurde keine Bewegungskompensation verwendet und die Parameter der Transformationen wurden nicht entropiekodiert. Es wurde außerdem nur jedes 3. Einzelbild kodiert, die ausgelassenen Einzelbilder werden am Dekodierer interpoliert. Bei einer Bitrate von 128 kBit/s (Kompressionsfaktor 154 bzw. 51,3 wenn man bedenkt, daß eigentlich nur jedes 3. Bild kodiert wird) ergaben sich SNR-Werte von 36-37 dB für die einzelnen Frames, bei 64 kBit/s (Kompressionsfaktor 308 bzw. 102,6) ergaben sich SNR-Werte von 32-36 dB (durchschnittlich etwa 34,5 dB) und bei 32 kBit/s SNR-Werte von 28-34dB (durchschnittlich etwa 31dB). Ali et al. berichten, daß sich bei letzterer Testsequenz sehr große Blockeffekte ergeben.

In [37] wurden Experimente mit den Testsequenzen "Miss America" und "Salesman" im QCIF-Format (176 × 144) bei einer Frame-Rate von 10 Hz gemacht. Bei einer Bitrate von 128 kBit/s (Kompressionsfaktor 15,84) ergab sich ein durchschnittlicher SNR-Wert von 37,4 dB mit "Miss America". Bei einer Bitrate von 192 kBit/s ergab sich mit "Salesman" ein durchschnittlicher SNR-Wert von 29,1 dB.

In [3] wurden keine Tests durchgeführt.

2.2.3 Vergleich der beiden Methoden

Der einzige Unterschied zwischen den beiden Methoden (Reine Inter-Frame und kombinierte Inter-/Intra-Frame-Kodierung) ist eigentlich nur, daß der Domain-pool (falls ein Block fraktal kodiert wird) einmal aus dem vorherigen Einzelbild genommen wird und einmal aus dem zu kodierenden Frame selbst genommen wird. Meiner Meinung nach dürfte das in der Bildqualität keine großen Unterschiede ausmachen, da die Domain-Blocks der beiden Domain-pools sehr ähnlich sind, aufgrund der Ähnlichkeit von aufeinanderfolgenden Frames. Wenn der Domain-pool aus dem

vorherigen Bild genommen wird, ist allerdings die Dekodierung schneller, da nicht iteriert werden muß . Die Auflösungsunabhängigkeit der kodierten Videos bleibt in beiden Fällen erhalten. Meiner Meinung nach ist also die erste Methode der zweiten vorzuziehen. Es hat übrigens wohl kaum Sinn, Domain-Blocks aus beiden Frames zu verwenden, da diese wohl sehr ähnlich sein dürften, und somit nur Bits zur Kodierung des Frames, aus dem die Domain-Blocks stammen, verschwendet werden.

2.2.4 Fraktale Videokodierung durch Bewegungsschätzung

In [30, 52, 53] wird ein etwas weiterentwickelter Ansatz vorgestellt, in dem versucht wird, die Parameter der Transformationen eines Frame anhand der Parameter des vorhergehenden Frames zu schätzen. Aufgrund der Ähnlichkeit zweier aufeinanderfolgender Frames der Sequenz sollten auch die Abbildungen ähnlich sein.

In [30] wird das Verfahren nur zur Beschleunigung des Suchalgorithmus eingesetzt, während in [52] und [53] das Verfahren auch zur Verbesserung des Verhältnisses Kompression zu Bildqualität verwendet wird. Ich werde mich im folgenden deshalb nur auf [52] und [53] beziehen.

Es wird zunächst zwischen "vorhersagbaren" (predicted) Einzelbildern und "nicht-vorhersagbaren" (non-predicted) Einzelbildern unterschieden. Nichtvorhersagbare Einzelbilder werden mit einer konventionellen fraktalen Standbildkompression kodiert. Paul und Hayes haben so in ihrer Implementation jedes 30. Einzelbild konventionell kodiert. Dadurch wird verhindert, daß Fehler mit der Zeit sehr stark ansteigen können. Zur Kodierung von vorhersagbaren Einzelbildern haben Paul und Hayes zwei Ansätze vorgeschlagen:

Ansatz I: Es wird angenommen, daß die Pixel in ihrer Helligkeit fast konstant bleiben in Richtung der Bewegung eines Objekts im Bild. Dies gibt den Anstoß, die IFS-Parameter eines Blocks des Frames f_i durch die IFS-Parameter eines Blocks im Frame f_{i-1} "vorherzusagen". Sei $R_{i-1,j}$ ein Range-Block des Frames f_{i-1}, der durch den Domain-Block $D_{i-1,j}$ approximiert wird. Sei weiterhin $R_{i,j}$ ein Range-Block des Frames f_i, der räumlich nah bei $R_{i-1,j}$ liegt. Um den Domain-Block $D_{i,j}$ zu finden, der $R_{i,j}$ möglichst gut approximiert, wird nur in einer kleinen Umgebung um den Domain-Block $D_{i-1,j}$ gesucht (siehe auch Abb.2.3) Typischerweise wählt man die Suchtiefe dabei so, daß der Abstand von $D_{i-1,j}$ zu $D_{i,j}$ maximal zwischen 7 und 16 Pixel in x- und y-Richtung ist. Dadurch wird einerseits die Bitanzahl, die nötig ist, um die Kodierung zu speichern, geringer und außerdem wird die Komplexität der Suche viel kleiner und die Kompression schneller. Paul und Hayes nehmen an, daß dieser Ansatz vor allem für mittlere Kompressionsraten geeignet ist.

Ansatz II: Dieser Ansatz ist nach den Autoren für sehr geringe Bitraten geeignet. Ein "vorhersagbares" Einzelbild wird wieder in Range-Blocks unterteilt. Ein Range-Block $R_{i,j}$ wird entweder fraktal oder als bewgungskompensierter Block kodiert. Sei $r'_{i,j}$ der Vektor aus den Pixeln des approximierten Range-Block $R'_{i,j}$. Die Approximation durch Bewegungskompensation wird dann gegeben durch:

$$r'_{i,j} = r_{i-\Delta i,j-1} + sd \tag{2.10}$$

Die Approximation wird also durch den Vektor $r_{i-\Delta i,j-1}$ eines Range-Block des letzten Frames kombiniert mit einem Domain-Block d, skaliert mit dem Faktor s, gegeben. Im Falle $s = 0$ wird einfach aus dem letzten Frame kopiert. Der Range-Block $R_{i-\Delta i,j-1}$ wird dabei aus einer kleineren Umgebung um $R_{i,j}$ genommen. Dadurch kann man so etwas wie Bewegungskompensation erreichen.

Es muß natürlich irgendwie entschieden werden, welche der beiden Approximationsmöglichkeiten verwendet werden. Paul und Hayes machen hier sehr vage

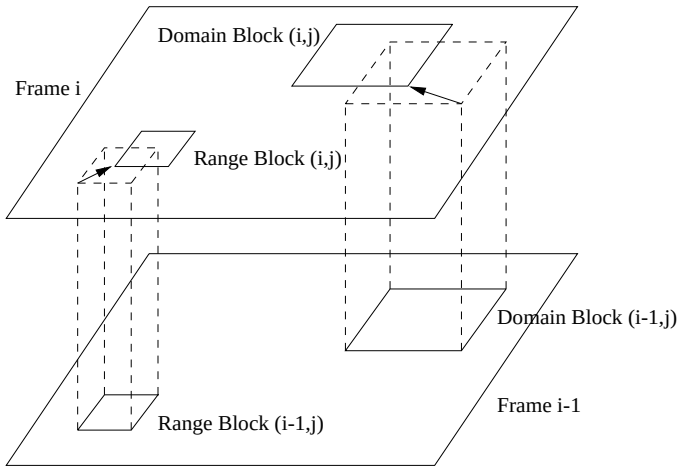

Abbildung 2.3: Domain-Block-Schätzung

Angaben. Sie sagen nur, daß diese Entscheidung adaptiv gemacht werden muß; je nachdem, ob eine Verringerung der Bildqualität mehr wiegt oder die Zunahme der benötigten Bitanzahl.

In [52] wurden Experimente mit den Testsequenzen "Miss America" und "Football" im Format 352×240 durchgeführt. Leider wurde seltsamerweise nichts über die erreichten Kompressionsraten gesagt und die Kompressionszeiten werden auch nicht erwähnt. Mit dem ersten Ansatz wurde jedenfalls ein mittlerer SNR-Wert von 37,91 dB mit der Testsequenz "Miss America" erreicht und mit der Testsequenz "Football" ein Wert von 28 dB. Mit dem zweiten Ansatz wurde 38,37 dB für "Miss America" erreicht.

In [53] wurden nur mit dem zweiten Ansatz Experimente durchgeführt. Es wurde die Testsequenz "Miss America" im Format 176×144 verwendet. Es wurde ein Bitrate von 112 kBit/s bei einer Frame-Rate von 30 Hz (Kompressionsfaktor 54,31) erreicht. Der mittlere SNR-Wert war 35 dB.

2.3 Videokodierung mit der Bath Fractal Transform

Monro et al. stellen den wohl am weitesten entwickelten fraktalen Videokodierer in [45, 46, 63, 50] vor. Sie verwenden die von ihnen entwickelte Bath Fractal Transform (siehe Kapitel 1.8), um Einzelbilder einer Farbvideo-Sequenz zu kodieren. Dabei nutzen sie zusätzlich Ähnlichkeiten aufeinanderfolgender Bilder aus. Die BFT hat dabei den Vorteil, daß die Kodierung im Gegensatz zu konventionellen fraktalen Schemata sehr schnell ist (selbst im Vergleich mit anderen Kompressionstechniken) und auch die Dekompression sehr schnell berechnet werden kann. So kann allein auf Software-Basis ein Kodierer und Dekodierer implementiert werden, der Echtzeitanwendungen zuläßt. Ein System, das sich für Anwendungen wie Bildtelefon eignet, ist in der Enwicklung. Monro et al. haben dafür einen Patentantrag eingereicht.

Ich berichte im folgenden über die aktuellste Beschreibung des Kodierers [50].

2.3.1 Funktionsweise des Kodierers

Ein Frame wird in Einzelbilder der YUV-Komponenten des Farbraumes zerlegt. Bei der Kodierung wird zunächst ein Differenzbild aus dem zu kodierenden und dem vorherigen dekodierten Frame berechnet. Dieses Differenzbild wird in sogenannte "super parent blocks" der Größe 32×32 Pixel eingeteilt. (Das Video sollte deshalb ein Format mit Vielfachen von 32 haben.) Diese "super parent blocks" werden dann nach einem Quadtree-Schema weiter aufgeteilt, wobei die das quadratische Mittel der Differenzwerte in den Quadraten als Unterteilungskriterium dient. Dies hat den Effekt, daß in Bereichen mit großer Bewegung die Quadrate kleiner sind und in Bereichen mit wenig Bewegung größer. Die kleinste erlaubte Größe der Blöcke ist 4×4 Pixel. Je nach erwünschter Bitrate wird in mehr oder weniger Blöcke aufgeteilt.

Nun werden die Blöcke nach der Größe des quadratischen Mittels der Differenzwerte geordnet und in dieser Reihenfolge kodiert. Dazu wird eine BFT der Ordnung 2 angewendet (siehe Kapitel 1.8). Die Parameter der Transformation werden berechnet, quantisiert, entropiekodiert und dem Dekodierer übermittelt. Wenn die gewünschte Bitanzahl erreicht ist, durch Anwendung dieses Prozesses auf die nachfolgenden Blöcke, wird die nächste Komponente des YUV-Farbraumes kodiert. Für die Y-Komponente werden 80 % der Bitrate verwendet und für die U- und V-Farbkomponenten jeweils 10 %. Die U- und V-Bilder werden vor der Kodierung jeweils in x- und y-Richtung durch Pixelmittelung um den Faktor 2 verkleinert. Wenn alle YUV-Einzelbilder kodiert sind, folgt das nächste Bild der Sequenz.

Durch diesen Prozeß werden die Teile des Bildes, in denen wenig Bewegung vorhanden ist, nicht dem Dekodierer übermittelt, da die Blöcke mit absteigender "Größe an Bewegung" kodiert werden und bei einer bestimmten Bitrate abgebrochen wird. Diese Teile werden einfach aus dem vorherigen Frame kopiert. Dadurch ist es leicht möglich, beliebige Bitraten zu erreichen

Um die visuelle Qualität zu erhöhen, schlagen Monro und Nicholls vor, von Zeit zu Zeit doch Teile des nicht sehr bewegten Hintergrundes zu übertragen. Dazu wird eine Tabelle angelegt, die den Zeitpunkt von jedem 4×4 Block beinhaltet, wann dieser zuletzt aufgefrischt wurde. Es wird dann zusätzlich zu den Blöcken mit großer Bewegung, der Block übertragen, der am "ältesten" ist. Diese Blöcke werden mit etwas niedriger Kompression bzw. höherer Bildqualität kodiert. Wie oft dies geschehen soll, wird allerdings nicht genau beschrieben. Es ist aber wohl sinnvoll, mit jedem Frame auch 1 oder 2 solcher Blöcke zu übertragen.

Dadurch, daß auch das erste Bild auf diese Weise kodiert wird, ist bei den ersten Bildern der Sequenz der Approximationsfehler noch groß . Dieses Problem spielt bei Anwendungen wie Bildtelefon aber wohl eine geringe Rolle, da dieser überdies schnell geringer wird. Außerdem ist das Problem auch leicht damit zu lösen, indem man das erste Bild vollständig kodiert (man hat dann eben für das erste Bild eventuell eine niedrigere Kompression).

Als Variation könnte man auch zusätzlich DCT-Kodierung verwenden. Und zwar wird hierzu geprüft, ob sich ein Block besser mit einer BFT der Ordnung 2 kodieren läßt oder mit einer DCT.

2.3.2 Ergebnisse

Es wurden Experimente mit der Testsequenz "Salesman" im Format 160×128 (Bildfrequenz 25 Hz) und verschiedenen Bitraten gemacht. Es wird leider nicht genau beschrieben, ob auch DCT-Kodierung verwendet wurde und wie die Quantisierung der Koeffizienten der BFT durchgeführt wird. Es ist allerdings zu vermuten, daß letzteres wie bei der Standbildkompression mit der BFT funktioniert (siehe Kapitel 1.8).

Bei einer Bitrate von 200 kBit/s (Kompressionsfaktor 69,12) hat sich ein durchschnittlicher SNR-Wert von etwa 33 dB, bei 100 kBit/s (Kompressionsfaktor 138,24) ein SNR-Wert von etwa 31 dB, bei 50 kBits/s (Kompressionsfaktor 276,48) ein durchschnittlicher SNR-Wert von etwa 28 dB, und bei 20 kBit/s (Kompressionsfaktor 691,2) ein durchschnittlicher SNR-Wert von etwa 26 dB ergeben.

2.4 Vergleich der verschiedenen Verfahren

Da ich nur einen der beschriebenen Codecs (siehe Kapitel 2.2.1) zu Verfügung hatte, konnte ich keine eigenen Vergleiche anstellen. Deshalb sind die Vergleiche nur auf die in den Papers angegebenen Kompressionsraten und SNR-Werte für die Testesequenz "Miss America" gestützt. Unglücklicherweise sind in allen Forschungsarbeiten verschiedene Formate der Sequenz verwendet worden, so daß der Vergleich noch schwieriger wird. In [39] konnte außerdem nur ein ungefährer Kompressionsfaktor angegeben werden, da die Quantisierung der Parameter noch nicht untersucht wurde. Es können hier also nur vage Aussagen gemacht werden zu der Leistungsfähigkeit der einzelnen Verfahren.

Tabelle 2.1 listet die in den einzelnen Artikeln berichteten Kompressionsfaktoren und SNR-Werte noch einmal auf. Außerdem sind Kompressionsraten und SNR-Werte für einen H.263-Kodierer (DCT-basiert) angegeben. Es ist wohl sinnvoll, nur Tests, bei denen das Bildformat ungefähr übereinstimmt, zu vergleichen. Es ist zu entnehmen, daß die in Kapitel 2.2.2 beschriebene kombinierte Inter-/Intra-Frame-Kodierung bessere SNR-Werte liefert als das in Kapitel 2.1.1 beschriebene 3-D-Videokodierverfahren. Die Kodierung eines Videos durch 3-D-Transformationen scheint also der Inter-/Intra-Frame-Kodierung unterlegen zu sein. Die Videokodierung mit der Bath-Fractal-Transform (Kapitel 2.3) scheint bei niedrigen Kompressionsraten mit Fisher's Interframe-Kodierer (Kapitel 2.2.1) gleich auf zu liegen, bei höheren Kompressionsraten jedoch zu unterliegen. Die Kodierung durch Bewegungsschätzung von Paul und Hayes (Kapitel 2.2.4,[52]) scheint wiederum ein bißchen besser wie Fisher's Inter-Frame-Kodierer zu sein. Insgesamt scheint die Kodierung durch Bewegungsschätzung von den verglichen Methode die beste zu sein. Um dies mit Sicherheit sagen zu können, sind allerdings gezielte Untersuchungen nötig. Im Vergleich mit H.263 schneiden alle fraktalen Videokodierverfahren deutlich schlechter ab.

Da Barthel et al. leider andere Testsequenzen zur Kodierung mit der von ihnen entwickelten kombinierten Fraktal-Transformations-Kodierung verwendet haben, ließ sich kein Vergleich anstellen. Diese Methode ist auch recht vielversprechend, da sie bei der Standbildkompression schon recht gute Resultate zeigt.

Literaturquelle, Kapitel	**Format der Testsequenz**	**Kompressionsfaktor**	**SNR-Wert (dB)**
[36], 2.1.1	360 × 288	74,39	≈ 33,5
		≈ 110	≈ 31,5
[39][a], 2.1.2	256 × 256	≈ 13	37,63
		≈ 25	36,22
		≈ 51	27,21
[24][b], 2.2.1	160 × 128	24,95	37,01
		42,31	35,74
		55,21	33,34
		74,20	33,73
		99,27	32,70
		124,69	32,04
		244,29	31,44
[34], 2.2.2	352 × 280	51,3 (128 kBit/s)	≈ 36, 5
		102,6 (64 kBit/s)	≈ 34, 5
		154 (32 kBit/s)	≈ 31
[37], 2.2.2	176 × 144	15,84	37,4
[53], 2.2.4	176 × 144	54,31 (112 kBit/s)	≈ 35
[50], 2.3	160 × 128	69,12 (200 kBit/s)	≈ 33
		138,24 (100 kBit/s)	≈ 31
		276,48 (50 kBit/s)	≈ 28
		691,2 (20 kBit/s)	≈ 26
H.263	352 × 288	293 (69 kBit/s)	38,06
		169 (120 kBit/s)	39,56
		84,5 (240 kBit/s)	40,68

[a] Da die Quantisierung der Transformationskoeffizienten noch nicht untersucht wurde, kann nur ein ungefährer Kompressionsfaktor angegeben werden (1 Byte pro Koeffizient)

[b] Die Kompressionsraten wurden teilweise mit verschiedenen Parametereinstellungen erreicht; es werden nur die besten der in [24] angegeben Testläufe aufgeführt.

Tabelle 2.1: Resultate der in der Literatur angegeben Tests mit fraktalen Video-Codecs

Schluß

Die fraktale Bildkodierung hat sich als wirkliche Konkurrenz zu konventionellen Kodierverfahren wie JPEG entwickelt. Vor allem bei hohen Kompressionsraten ist die erreichbare Bildqualität deutlich besser. Sie hat allerdings den Nachteil, daß die benötigtem Zeiten zur Kompression eines Bildes recht hoch liegen, wobei die Dekompressionszeiten allerdings ungefähr denen der JPEG-Kompression entsprechen. Die Anwendungsgebiete der fraktalen Kompression liegen daher zur Zeit wohl mehr im Bereich Datenarchivierung, bei der einmal komprimiert wird und öfters dekomprimiert wird. Da die Computer aber immer schneller werden, könnte die Rechenzeit in Zukunft vielleicht auch kein Problem mehr darstellen.

Ein sehr interessanter Nebeneffekt der fraktalen Bildkompression ist die Auflösungsunabhängigkeit des kodierten Bildes. Die Möglichkeit, Bilder beliebig zu skalieren, die keine andere Kompressionsmethode bietet, dürfte für viele Anwendungen interessant sein.

Es steht zu erwarten, daß die fraktale Kompression weiter zu verbessern ist. Es wurde schließlich erst seit 7 Jahren an dieser neuen Technik entwickelt. Wenn man diesen Aufwand vergleicht mit dem, der an der Entwicklung von JPEG betrieben wurde, ist dies ein kleiner Bruchteil. Vor allem von Hybridtechniken wie kombinierter Fraktal-DCT-Kompression oder Fraktal-Wavelet-Kompression ist noch einiges an Fortschritt zu erwarten.

Die fraktale Videokompression ist noch deutlich weniger weit fortgeschritten wie die fraktale Standbildkompression. Die erzielten Resultate sind doch noch schlechter als die mit Standardkompressionsalgorithmen wie MPEG oder H.263 erreichbaren. Da auf dem Gebiet der fraktalen Videokompression noch relativ wenig geforscht wurde, steht zu erwarten, daß weitere Entwicklung diese Methode noch verbessern wird und vielleicht erreicht oder übertrifft diese dann auch die Leistungsfähigkeit der Standardvideokompressionstechniken.

Danksagungen

Ich danke der Siemens AG, bei der ich im Laufe meiner Werkstudententätigkeit diesen Bericht angefertigt habe, insbesondere Thomas Riegel[1] und Dr. Kutka, die mich betreut haben. Außerdem danke ich der Stiftung Jugend-forscht. Der Wettbewerb war mir ein großer Ansporn, auf dem Gebiet der fraktalen Kompression Forschung und Entwicklung zu betreiben. Weiterhin bedanke ich mich bei der Eduard-Rhein-Stiftung für Informationstechnik, dessen Jugendpreis ich im Rahmen meines Jugend-forscht-Projektes erhalten habe.

[1] email: Thomas.Riegel@mchp.siemens.de

Literaturverzeichnis

[1] M. Ali, T.G. Clarkson, *Using linear fractal interpolation functions to compress static and motion images*, *Proc. 3rd Conf. on Information Technology and its Applications (ITA'94)*

[2] M. Ali, T. Clarkson, *Using linear fractal interpolation functions to compress video images*, Fractals 2,3 (1994) 417-421.

[3] M. Ali, C. Papadopoulos, T. Clarkson, *The use of fractal theory in a video compression system*, in: *Proceedings DCC'92 Data Compression Comference*, J.A. Storer and M. Cohn (eds.), IEEE Comp. Soc. Press. 1992.

[4] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A. Wu, *An Optimal Algorithm for Approximate Nearest Neighbor Searching*, in: *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, New York 1994

[5] M. Barnsley, L. Hurd, M. Gustavus, *Fractal video compression*, in: *Proceedings of the Thirty-Seventh IEEE Computer Society International Conference*, Vol. 37, pp. 41-42, 1992.

[6] Z. Baharav, D. Malah, E. Karnin, *Hierarchical interpretation of fractal image coding and its application to fast decoding*, in: *Int. Conf. on Digital Signal Processing*, Zypern, Juli 1993.

[7] B. Bani-Eqbal, *Combining Tree and Feature Classification in Fractal Encoding of Images*, Manchester, 1997.

[8] K.U. Barthel, *Entropy constrained fractal image coding*, in: *NATO ASI Conf. Fractal Image Encoding and Analysis*, Trondheim, Juli 1995.

[9] K.U. Barthel, T. Voyé, *Adaptive fractal image coding in the frequency domain*, in: *Proc. ICIP-94 IEEE International Conference on Image Processing: Theory, Methodology, Systems and Applications*, Budapest, Juni 1994.

[10] M.F. Barnsley, A. Sloan, *Methods and Apparatus for image compression by iterated function system*, United States Patent # 4,941,193, 1990.

[11] M.F. Barnsley, A. Sloan, *Method and Apparatus for processing digital data*, United States Patent # 5,065,447, 1991.

[12] K.U. Barthel, J. Schüttemeyer, T. Voyé, P. Noll, *A new image coding technique unifying fractal and transform coding*, in: *Proc. ICIP-94 IEEE International Conference on Image Processing*, Austin, Texas, Nov. 1994.

[13] K.U. Barthel, T. Voyé, *Three-dimensional fractal video coding*, in: *Proc. ICIP-95 IEEE International Conference on Image Processing*, Washington, D.C., 1995.

[14] K.U. Barthel, T. Voyé, *Combining wavelet and fractal coding for 3-D video coding*, in: *Proc. ICIP-96 IEEE International Conference on Image Processing*, Lausanne, Sept. 1996.

[15] A. Bogdan, *Multiscale (inter/intra-frame) fractal video coding*, in: *Proc. ICIP-94 IEEE International Conference on Image Processing*, Austin, Texas, Nov. 1994.

[16] R.D. Boss, E.W. Jacobs, *Archetype classification in an iterated teransformation image compression algorithm*, in [25], S. 79-90.

[17] G. Caso, P. Obrador, C.-C.J. Kuo, *Fast methods for fractal image encoding*, in: *Proceedings from IS& T/SPIE 1995 Symposium on Electronical Imaging: Science & Technology*, Vol. 2501, S. 583-594, 1995.

[18] G. Ciscar, *On entropy coding Fisher's Fractal quadtree code*, Juni 1996

[19] K.M. Cheung, M. Shahshahani, *A comparison of the fratal and JPEG algorithms*, TDA Progress Report 42, 107 (1991) 21-26

[20] F. Davoine, J.-M. Chassery, *Adaptive Delaunay Trinagulation for Attractor Image Coding* in: *Proc. of 12th International Conference on Pattern Recognition (IPCR)*, Jerusalem, Oct. 1994.

[21] F. Davoine, J. Svensson, J.M. Chassery, *A mixed triangular and quadrilateral partition for fractal video coding*, in: *Proc. ICIP-95 IEEE International Conference on Image Processing*, Washington, D.C., 1995.

[22] G. Davis, *Adaptive Self-Quantization of Wavelet-Subtrees: A wavelet-based thoery of Fractal Image Compression*, SPIE Conf. on Mathematical Imaging: Wavelet applications in Signal and Image Processing, San Diego, Juli 95.

[23] G.J. Ewing, C.J.Woodruff, *Comparison of JPEG and fractal-based image compression on target acqisition by human observer*, Optical Engineering 35 (1996) 284-288

[24] Y. Fisher, D. Rogovin, T.-P. Shen, *Fractal (self-VQ) encoding of video sequences*, in: *Proceedings from SPIE Visual Communications and Image Processing*, Chicago, Sept. 1994.

[25] Y. Fisher (ed), *Fractal Image Compression, Theory and Application*, Springer Verlag, New York, 1995.

[26] Y. Fisher, *Fractal Image Compression with Quadtrees*, in: [25], S. 55-77.

[27] Y. Fisher, S. Menlove, *Fractal Encoding with HV Partitions*, in: [25], S. 119-136.

[28] Y. Fisher, T.P. Shen, D. Rogovin, *A comparison of fractal methods with dct (jpeq) and wavelets (epic)*, in: *Proceedings from SPIE Neural and Stochastic Methods in Image and Signal Processing*, Vol. 2304-16, 1994

[29] J.H. Friedman, J.L. Bentley, R.A. Finkel, *An Algorithm for finding best Matches in Logarithmic Expected Time*, in: ACM Transactions on Mathematical Software, Vol. 3, No. 3, September 1977

[30] R.E.H. Franich, R.L. Lagendijk, J. Biemond, *Fractal picture sequence coding: Looking for the effective search* in: *Proceedings of the International Picture Coding Symposium PCS'94*, Sacramento, California, Sept. 1994.

[31] M. Gharavi-Alkhansari, T. Huang, *Fractal video coding by matching pursuit*, in: *Proceedings ICIP-96 IEEE International Conference on Image Processing*, Lausanne, Sept. 1996.

[32] M. Gharavi-Alkhansari, T. Huang, *Fractal-based image and video Coding*, in: *Video Coding: The Second Generation Approach*, L. Torres und M. Kunt (eds.), Kluwer Academic Publishers, Boston, MA, 1996.

[33] R. Hamzaoui, *Codebook clustering by self-organizing maps for fractal image compression*, in: *NATO ASI Conf. Fractal Image Encoding and Analysis*, Trondheim, Juli 1995.

[34] B. Hürtgen, P. Büttgen, *Fractal approach to low-rate video coding*, in: *Proceedings from SPIE Visual Communications and Image Processing*, Vol. 2094, pp. 120-131, 1993.

[35] J.D. Johnston, *A Filter Family Designed for Use in Quadrature Mirror Filter Banks*, in: *Proc. 1980 IEEE Int. Conf. Acoust. Speech Signal Process., pp. 291-294* April, 1980.

[36] M.S. Lazar, L.T. Bruton, *Fractal block coding of digital video*, IEEE Trans. on Circuits and Systems for Video Technology 4,3 (1994) 297-308

[37] J. Liu, S. Marlow, N. Murphy, *Multilevel fractal block coding in video compression*, in: *Proceedings of Amsterdam Conference "1993 DSP - The Enabling Technology for Communication*, 1993.

[38] S.P. Lloyd, *Least Squares Quantization in PCM*, in: *IEEE Trans. on Information Theory, Vol.IT-28. No.2*, März, 1982.

[39] H. Li, M. Novak, R. Forchheimer, *Fractal-based image sequence compression scheme*, Optical Engineering 32,7 (1993) 1588-1595.

[40] H. Lin, A.N. Venetsanopoulos, *Fast fractal image coding using pyramids*, in: *Proceedings of 8th International Conference on Electrical and Computer Engineering*, Montreal, Sept. 1995.

[41] S.G. Mallat, *A Theory for Multiresolution Signal Decomposition: The Wavelet Represantation*, in: *IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.11, no. 7,pp. 674-693*, Juli 1989.

[42] B. Mandelbrot, *The Fractal geonetry of Nature*, San Francisco, Freeman, 1982.

[43] D.M. Monro, *A hybrid fractal transform*, in: *Proceedings of ICASSP-1993 IEEE International Conference of Aoustics, Speech and Signal Processing*, Vol. 5, pp. 169-172, 1993

[44] D. Monro, F. Dudbridge *Fractal Approximation of image blocks* in: *Proceedings of ICASSP-1992 IEEE International Conference on Acoustics, Speech and Signal Processing* Vol. 3, pp. 485-488, 1992

[45] D.M. Monro, J.A. Nicholls, *Real time fractal video for personal communication* Fractals 2,3 (1994) 391-394.

[46] D.M. Monro, J.A. Nicholls, *Low bit rate fractal video, Proc. ICIP-95 IEEE International Conference on Image Processing*, Washington, D.C., 1995.

[47] D.M. Monro, D.L. Wilson, J.A. Nicholls, *High speed image coding with the Bath fractal transform*, in: *IEEE International Symposium on Multimedia Technologies and Future Applications*, Southampton, April 1994

[48] D. Monro, S.J. Woolley *Fractal Image Compression without searching* in: *Proceedings of ICASSP-1994 IEEE International Conference on Acoustics Speech and Signal Processing* Vol. 5, Adelaide, 1994.

[49] M. Nelson, *The Data Compression Book*, M&T Books, New York, 1996.

[50] J.A. Nicholls, D.M. Monro, *Scalable video by software*, in: *Proc. ICASSP 1996*, Atlanta, 1996.

[51] M. Novak, *Attractor coding of images*, Licentiate Dissertation, Dept. of Electrical Engineering, Linköping University, Mai 1993.

[52] B.-B. Paul, M.H. Hayes, *Fractal-based compression of motion video sequences*, in: *Proc. ICIP-94 IEEE International Comference on Image Processing*, pp. 755-759, Austin, Texas, Nov. 1994.

[53] B.-B. Paul, M.H. Hayes, *Video Coding based on Iterated function systems*, in: *Proceedings of ICASSP-1995 IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, 1995.

[54] R. Rinaldo, G. Calvagno, *Image Coding by block prediction of multiresolution subimages*, *IEEE Transactions on Image Processing*, 4(7), Seiten 909-920, Juli 1995.

[55] D. Saupe, *Breaking the time complexity of fractal image compression*, Technical Report 53, Institut für Informatik, Universität Freiburg, 1994

[56] D. Saupe, *From classification to multidimensional keys*, in: *Fractal Image Compression - Theory and Application*, Y. Fisher (ed.), Springer Verlag, New York, 1994.

[57] D. Saupe, *Accelerating fractal image compression by multi-dimensional nearest neighbor search*, in: *Proceedings DCC'95 Data Compression Conference*, J. A. Storer and M. Cohn (eds), IEEE Comp. Soc. Press., March 1995.

[58] D. Saupe, *Fractal Image Compression via nearest neighbor search*, in: Conf. Proc. NATO ASI Fractal Image Encoding and Analysis, Trondheim, July 1995, Y. Fisher (ed), Springer Verlag. New York, 1995.

[59] D. Saupe, R. Hamzaoui, *Complexity reduction methods for fractal image compression*, in: *I.M.A. Conf. Proc. on Image Compression; Mathematical Methods and Applications*, Sept. 1994, J.M. Blacklegde (ed), Oxford University Press, 1996.

[60] D. Saupe, H. Hartenstein, *Lossless acceleration of fractal image compression by fast convolution*, in: *Proc. ICIP-96 IEEE International Conference on Image Processing*, Lausanne, Sept. 1996.

[61] J.M. Shapiro, *Embedded image coding using zerotrees of wavelet-coefficients*, *IEEE Transactions on Signal Processing*, 41,(12), Seiten 3445-3462, Dezember 1993.

[62] D.F. Watson, *Computing the n-dimensional Delaunay tesselation with application to Voronoi polytops*, Comput. J., 24(2):167-172, 1981.

[63] D.L. Wilson, J.A. Nicholls, D.M. Monro, *Rate buffered fractal video*, in: *Proceedings of ICASSP-1994 IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, 1994.

[64] S.J. Woolley, D.M. Monro, *Optimum parameters for hybrid fractal image coding*, in: *Proceedings of ICASSP-1995 IEEE International Conference on Acoustics Speech and Signal Processing*, Detroit, 1995.

[65] Z. Xiong, K. Ramchandran, M.T. Orchard, K. Asai, *Wavelet packets-based image coding using joint space-frequency quantization*, in: *Proceedings of IEEE International Conference on Image Processing*, volume 3, Seite, 324-328, Austin, Texas, Nov. 13-16, 1994.