Mechanismen zur Übertragung und zur platzsparenden interaktiven Darstellung grosser Bilddateien in mobilen Umgebungen

Studienarbeit
von
Tino Weinkauf

14. Juli 1999

Betreuer:
Prof. H. Schumann
Dipl.-Inf. U. Rauschenbach

Universität Rostock
Fachbereich Informatik
Institut für Computergraphik

Es ist die Zeit
Manchmal zu sehen
Oder zu hören
Was Du eben
Gerade schon
Ersonnen hast

Du kommst zu spät

kaspar loob

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation und Überblick

Information ist das Schlüsselwort einer modernen Gesellschaft. Neben dem bloßen Besitz kommt der Verarbeitung und Verknüpfung von Informationen eine besonders wichtige Rolle zu.

Oft wird Information visualisiert, weil davon ausgegangen werden kann, dass sich Zusammenhänge in Bildern dem menschlichen Denken eher erschließen als in langen Zahlenkolonnen. Eine große Bedeutung kommt der Verfügbarkeit dieser Bilder und Grafiken zu, denn nur so können sie am jeweiligen Ort zur jeweiligen Zeit ihre Verwendung finden. Es ist also insbesondere in der heutigen Welt der globalen Kommunikation wichtig, Mechanismen zu erforschen, die Bilder schnellstmöglich übertragen und leicht verständlich präsentieren.

Die vorliegende Arbeit beschäftigt sich mit diesem Problem im besonderen Bezug auf Umgebungen, die zur Übertragung und Darstellung von Bildern nur begrenzte Ressourcen bieten. Dieser Kontext fordert eine effiziente Komprimierung der Bilddaten sowie deren nichtredundante Übertragung und platzsparende Darstellung.

Uwe Rauschenbach stellt in seinen Arbeiten zur progressiven Bildübertragung und flexiblen Bildkommunikation Techniken zur Komprimierung von Rasterbilddaten mittels Wavelets vor und entwickelt ein Protokoll zur nichtredundanten, flexiblen Übertragung der komprimierten Datenströme ([Rau98], [RS98]). Im engen Zusammenhang dazu fokussiert diese Arbeit auf die Verwaltung der übertragenen Bilddaten und deren platzsparende, interaktive Darstellung.

Kapitel 2 stellt eine Technik vor, die in der Lage ist, große Bilddatenmengen virtuell zu verwalten. Damit wird eine lokale Zwischenspeicherung bereits übertragener Daten unter geringstmöglicher Beanspruchung der Ressourcen ermöglicht und somit die redundanzfreie Übertragung unterstützt. Eine Zugriffsoptimierung verringert die Anforderungszeiten für bereits lokal gespeicherte Bilddaten und ermöglicht somit der anfordernden Anzeigetechnik eine schnelle Darstellung. Dies ist insbesondere während der Ausführung von Interaktionen sehr wichtig.

Kapitel 3 widmet sich dem rechteckigen FishEye–View. Diese Anzeigetechnik ermöglicht es mit Hilfe eines Fokus-und-Kontext Ansatzes, große Rasterbilder auf kleinen Anzeigeflächen darzustellen und interaktiv zu explorieren.

Die praktische Umsetzung der Kommunikation zwischen dem Bildserver und dem Bildclient wird im Kapitel 4 vorgestellt. Hierbei geht es um die Initialisierung und Aufrechterhaltung der Kommunikation mittels Steuerinformationen. Es wird somit ein Transportmechanismus für die Datenströme der durch Uwe Rauschenbach entwickelten Übertragungstechnik vorgestellt.

Den Abschluss der Arbeit bildet Kapitel 5, in dem eine Zusammenfassung und ein Ausblick geboten werden.

1.2 Begriffswelt

In diesem Kapitel sollen die wichtigsten, grundlegenden Begriffe und Abkürzungen der vorliegenden Arbeit definiert werden. Dies dient sowohl dem Verständnis als auch der weiteren Einführung in den zugrundeliegenden Kontext.

Level of Detail (LoD): Ein (lokaler[1]) LEVEL OF DETAIL ist eine Menge von Vektoren in einem Vektorraum V, der durch die vier unabhängigen Dimensionen Auflösung in x- bzw. y-Richtung, Farbe und Präzision aufgespannt wird. Der LoD ist also ein Maß für die Detailierung von Bilddaten. Synonym dazu wird auch der Begriff DETAILSTUFE benutzt. Die in dieser Arbeit vorgestellten Komponenten nutzen dieses Konzept in erster Linie in Bezug auf die Auflösungen in x- bzw. y-Richtung. Uwe Rauschenbach und Heidrun Schumann stellen in [RS98] eine formale Definition des LoD vor.

Region of Interest (RoI): Eine REGION OF INTEREST ist definiert durch eine Menge von zusammenhängenden Pixeln (o.B.d.A. ein Polygon[2]) und zwei (lokalen) LEVELS OF DETAIL: dem Übertragungsziel und dem Übertragungszustand. Eine REGION OF INTEREST definiert also einen Bildausschnitt in einer bestimmten (zu erreichenden bzw. bereits erreichten) Detailstufe. Eine formale Definition findet sich in [RS98].

Differentielle Übertragungstechnik: Da für die LEVELS OF DETAIL ein Vektorraum definiert ist, lassen sich zwei LoD zu einem höheren LoD in bestimmter Weise addieren. Eine differentielle Übertragungstechnik unterstützt dieses Konzept derart, dass zur Erreichung eines höheren LEVEL OF DETAIL die bereits übertragenen Daten im niedrigeren LoD genutzt und nur die der Differenzvektormenge entsprechenden Bilddaten übertagen werden. Dies ist der wesentliche Ansatz zur Erreichung einer redundanzfreien Übertragung.

Auflösungszahl: Die Auflösung von Bilddaten wird getrennt für die x- bzw. y-Richtung angegeben. Die Auflösungszahl ist dabei der ganzzahlige Divisor zur Skalierung der Bilddaten in die jeweilige Richtung. Eine Auflösungszahl n

[1] Auf globale LEVEL OF DETAIL wird im Rahmen dieser Arbeit nicht eingegangen. Siehe dazu [RS98].

[2] Unter der Maßgabe von Rasterbilddaten können alle Mengen von zusammenhängenden Pixeln als Polygon beschrieben werden.

eines Bildausschnittes für eine Richtung besagt also, dass der vorliegende Ausschnitt n-mal kleiner als der entsprechende voll aufgelöste Bildausschnitt ist. Die Auflösung von Bilddaten wird demnach durch die beiden mittels Doppelpunkt getrennten Auflösungszahlen für die jeweilige Richtung angegeben:[3]

$$\text{Auflösungszahl in } x : \text{Auflösungszahl in } y$$

Bilddaten mit höherzahligen Auflösungen sind somit niedriger aufgelöst als Bilddaten mit niedrigen Auflösungszahlen.

Pixelquadratische Auflösung: Im Rahmen dieser Arbeit wird von einer pixelquadratischen Auflösung gesprochen, wenn die Auflösungszahlen für beide Richtungen gleich sind.

2er-potenzpixelquadratische Auflösung: Bilddaten haben eine 2er-potenzpixelquadratische Auflösung genau dann, wenn

1. sie eine pixelquadratische Auflösung besitzen und
2. ihre Auflösungszahlen für beide Richtungen Potenzen von 2 sind.

Es handelt sich also um pixelquadratische Auflösungen mit 2er-Potenz–Auflösungszahlen.

Teilbild: Ein Teilbild ist ein rechteckiger Bildausschnitt. Synonym dazu wird auch der Begiff Bildteil verwendet.

Common Gateway Interface (CGI): CGI ist eine Schnittstelle für WWW–Server, die es ermöglicht, auf dem Server Programme (sogenannte CGI-Scripts) aufzurufen, diesen bestimmte Parameter zu übergeben und die vom Programm erzeugten Daten an den aufrufenden Client zu übertragen. Bei jedem Aufruf wird eine separate Instanz des Programmes erzeugt. CGI ist auf allen relevanten WWW–Servern verfügbar.

[3] **Beispiel:** Sei ein Bild (oder ein Bildausschnitt) in der Auflösung 2:4 vorhanden. Im Vergleich zum voll aufgelösten Bild (bzw. Bildausschnitt) ist es entlang der x-Achse nur halb so groß und entlang der y-Achse nimmt es sogar nur ein Viertel der eigentlichen Bildgröße ein.

Kapitel 2

Die lokale Bilddatenverwaltung

2.1 Motivation

In der Informationsgesellschaft der Gegenwart werden umfangreiche Datenströme produziert, verarbeitet und übertragen. Jedoch sind die Ressourcen für Speicherung, Verarbeitung sowie Übertragung begrenzt, und deshalb werden intelligente Lösungen gesucht, die die Daten komprimieren, klassifizieren, visualisieren und schnellstmöglich zugänglich machen.

Besonders der Aspekt der rechnergestützten Kommunikation nimmt einen hohen Stellenwert in der Informationsgesellschaft ein. Da jedoch die Bandbreite solcher Kommunikationswege begrenzt ist, werden die zu übertragenden Daten oftmals vor der Versendung komprimiert. Zudem existieren Techniken, die es erlauben, Daten in verschiedenen Detailstufen anzufordern, um den Bedürfnissen unterschiedlicher Nutzer gerecht zu werden und gleichzeitig Bandbreite bzw. Übertragungszeit zu sparen. In diesem Zusammenhang sei auf die Arbeiten von Uwe Rauschenbach verwiesen ([Rau98]).

Sind die Daten erstmal auf dem lokalen System angekommen, so sollten dort Techniken existieren, die eine erneute Übertragung dieser Daten verhindern, wenn sie ein zweites Mal angefordert werden. Diese Techniken werden i.a. als *Caches* bezeichnet. Die Arbeit eines Cache erfolgt zumeist transparent, was bedeutet, dass der Nutzer seine Existenz nicht bemerkt. Im Cache vorhandene Daten werden sofort und unverändert zur Präsentation bereitgestellt, während nicht vorhandene Daten nachgeladen werden müssen. Aufgrund der Transparenz ist es möglich, mehrere Caches zu kaskadieren.

Die in diesem Kapitel vorgestellte Technik implementiert einen zweistufigen, lokalen Cache für Rasterbilddaten. Dieser Cache versteht sich als ein Bindeglied zwischen einer Übertragungstechnik für Rasterbilder (z.B. mittels Wavelets) und einer lokalen Anzeigetechnik (z.B. der im Kapitel 3 ab Seite 25 vorgestellte rechteckige FishEye–View). Es werden Bilddaten verschiedener Auflösungsstufen und Segmentierung von Bildern unterstützt. Besonders in diesen Bereichen liegen die Vorteile der lokalen Bilddatenverwaltung. Zudem erfordert jede Anzeigetechnik für Rasterbilder schnelle Antwortzeiten auf Datenanforderungen. Dies spiegelt sich im zweistufigen Aufbau der lokalen Bilddatenverwaltung aus Festplattencache und Speichercache wider.

2.2 Anforderungen

Wie bereits erwähnt, soll die lokale Bilddatenverwaltung als ein Bindeglied zwischen einer Übertragungstechnik und einer Anzeigetechnik funktionieren.[1] Beide Techniken müssen in ihren Funktionen weitestgehend unterstützt werden, damit die Bilddatenverwaltung nicht zum „Flaschenhals“ im gesamten System wird.

Es wird von einer Übertragungstechnik ausgegangen, die Levels of Detail und Regions of Interest unterstützt. Dementsprechend muss auch die lokale Bilddatenverwaltung die Möglichkeit bieten, Teilbilder in verschiedenen Detailstufen zu verwalten.

Verschiedene Anzeigetechniken sollen mit der Bilddatenverwaltung kombinierbar sein. Es wird davon ausgegangen, dass die Anzeigetechnik dem Nutzer gewisse Interaktionstechniken zur Manipulation der Darstellung bietet. In diesem Zusammenhang ist die Geschwindigkeit entscheidend, mit der die Bilddaten der Anzeigetechnik bereitgestellt werden können. Nur bei geringen Antwortzeiten der lokalen Bilddatenverwaltung auf Anforderungen ist die Darstellung während der Interaktion angemessen schnell zu aktualisieren. Hier sind Lösungen anzubieten, die an die verschiedenen Ansprüche unterschiedlicher Anzeigetechniken anpassbar sind. Zudem ist es auch hier notwendig, verschiedene Bildteile in verschiedenen Detailstufen zur Verfügung zu stellen, da gewisse Anzeigetechniken (wie etwa der im Kapitel 3 vorgestellte FishEye–View) dies benötigen.

Da der Clientrechner, auf dem die lokale Bilddatenverwaltung, die Anzeigetechnik und eventuell zusätzliche Prozesse ausgeführt werden, i.a. über geringe Ressourcen verfügt, sind Lösungen zu finden, die die Ressourcen des Clients möglichst gering belasten. Hierbei kommt ein in der Informationstechnik häufig vorkommender Konflikt zum tragen: Die Abwägung zwischen Rechenzeit und Speicherplatzbedarf. Entscheidet man sich für kurze Rechenzeiten, so muss man einen erhöhten Speicheraufwand in Kauf nehmen. Optimiert man den Speicherbedarf, erhöhen sich die notwendigen Rechenzeiten. Für die praktische Umsetzung und Verwendung der lokalen Bilddatenverwaltung ist ein Mittelweg zu finden, bei dem der Bedarf beider Ressourcen in Grenzen zu halten ist.

Die sich aus den obigen Spezifikationen und Überlegungen ergebenden Forderungen werden im folgenden aufgeführt.

Segmentierung der Bilddaten: Es müssen Teilbilder einzeln speicher- und abrufbar sein. Dies ist notwendig, um die Regions of Interest-Funktionalität der Übertragungstechnik zu unterstützen. Dies wiederum spart sowohl Speicherplatz auf dem lokalen Rechner als auch Bandbreite während der Übertragung.

Detailstufen für Bilddaten: Die Bilddaten müssen in verschiedenen Levels of Detail speicherbar sein. Dies ergibt sich aus den obigen Spezifikationen für Anzeige- und Übertragungstechnik.

[1] Dies ist bezogen auf Techniken für Rasterbilder.

Interpolation von Bilddaten höherer LoD aus niedrigeren LoD: Dieses Verfahren wird als „Fallback“ bezeichnet. Es beinhaltet, dass durch die Anzeigetechnik angeforderte, aber noch nicht in der angeforderten Detailstufe übertragene Bilddaten aus sich bereits in der Bilddatenverwaltung befindlichen Daten mit geringerer Detailstufe generiert werden können. Somit wird es ermöglicht, der Anzeigetechnik bereits dann Daten zu liefern, wenn die Übertragungstechnik diese noch nicht (im angefordeteten LoD) bereitgestellt hat. Der angeforderte LoD wird durch das Fallback *nicht* erreicht. Die Daten behalten den zur Generierung genutzten LoD, jedoch wird der Anzeigetechnik der höhere LoD vorgetäuscht.[2]

Generierung von Bilddaten niedriger LoD aus höheren LoD: Es ist ohne weiteres möglich, aus Bilddaten mit höherer Detailstufe Bilddaten mit geringerem Detailgrad zu erzeugen. Dies ermöglicht redundanzfreie Speicherung und verringert somit den Platzbedarf. Jedoch erfordert eine solche Technik auch eine hohe Rechenleistung und zudem müssen wesentlich mehr Daten aus der Bilddatenverwaltung gelesen werden.

Virtuelle Verwaltung: Da die Übertragungstechnik nur bestimmte Bildausschnitte überträgt, sollen auch nur die bereits übertragenen Bildteile physisch auf dem Client angelegt werden. Die fehlenden Bildteile werden virtuell verwaltet. Dies verringert den Platzbedarf auf dem Client und schont somit dessen Ressourcen.

Nutzerbestimmte Lebensdauer: Die Daten in der lokalen Bilddatenverwaltung sollten dem Nutzer nicht nur für eine Sitzung zur Verfügung stehen. Falls erneut auf dasselbe Bild zugegriffen wird, müssen die bereits übertragenen Teile nicht noch einmal mittels der Übertragungstechnik angefordert werden. Der Nutzer sollte über den Verbleib bzw. die Löschung der Cache–Daten auf dem Client selbst entscheiden können.

Unkomprimierte Bilddatenspeicherung: Um der Anzeigetechnik schnellstmöglich Daten zur Verfügung stellen zu können, sollten diese unkomprimiert auf dem Client gespeichert werden. Dies erhöht natürlich den Platzbedarf.

Zugriffsoptimierung für häufig abgefragte Bildteile: Bilddaten, die durch die Anzeigetechnik oft angefordert werden, sollten so verwaltet werden, dass sie schnellstmöglich bereitgestellt werden können. Auf dieses Verhalten sollte die Anzeigetechnik Einfluß haben.

Bis auf die Generierung von Bilddaten niedriger LoD aus höheren LoD erfüllt die in diesem Kapitel vorgestellte lokale Bilddatenverwaltung alle oben genannten Forderungen. Die Generierung niedrig detailierter Bilddaten aus höher detailierten wurde nicht umgesetzt, da dies zuviel Rechenleistung bedarf und somit die Anzeigetechnik nicht schnell genug bedient werden kann. Die daraus resultierende Redundanz wird im Abschnitt 2.3.4 auf Seite 17 untersucht.

[2] Im Bezug auf die Auflösung der Bilddaten kann der Fallback mittels Streckung entlang der Koordinatenachsen geschehen.

2.3 Konzepte und Betrachtungen

2.3.1 Konzept des Tiling

Durch die RoI–Unterstützung der Übertragunstechnik werden nur bestimmte Teile eines Bildes übertragen und somit liegen auch nur Ausschnitte des gesamten Bildes lokal vor. Es ist also notwendig, ein Konzept zu entwickeln, dass Bildteile verwalten kann, denn wie bereits im Abschnitt 2.2 dargelegt, muss die Verwaltung des *gesamten* Bildes aus Speicherplatzgründen virtuell erfolgen. Es gibt im Wesentlichen zwei Ansätze, dieses Ziel zu erreichen:

1. Die Bildteile werden in ihrer übertragenen Größe gespeichert und mit den zugehörigen Koordinaten verwaltet.
2. Es wird eine gleichmäßige Aufteilung des Gesamtbildes eingeführt und die übertragenen Bildteile werden entsprechend dieser Einteilung gespeichert und verwaltet.

Der Vorteil des ersten Ansatzes besteht darin, dass nur exakt soviel Bilddaten physisch angelegt werden wie benötigt. Jedoch ist hierbei der Verwaltungsaufwand nicht eindeutig berechenbar und die zu erwartende hohe Fragmentierung des Gesamtbildes erhöht die Antwortzeiten auf Anforderungen der Anzeigetechnik.[3]

Der zweite Ansatz wurde bei der lokalen Bilddatenverwaltung umgesetzt und ermöglicht eine Aufteilung des Gesamtbildes in kleine, gleichgroße Bildteile (sogenannte Patches bzw. Tiles). Wird von der Übertragungstechnik ein Bildausschnitt übertragen, so brauchen nur die von ihm überlappten Patches lokal gespeichert werden. Die Patches, die nicht von bereits übertragenen Bildausschnitten überlappt werden, werden nicht physisch angelegt, sondern nur virtuell verwaltet. Abbildung 2.1 verdeutlicht die Vorgehensweise.

Der Vorteil dieses Ansatzes besteht im konstanten Verwaltungsaufwand. Jedoch werden i.a. mehr Bilddaten physisch angelegt als eigentlich benötigt, was sich für eine größere Anzahl von angrenzenden Bildausschnitten allerdings wieder relativiert.

Wenn sich die Größe des Gesamtbildes nicht ganzzahlig durch die Patchgröße teilen läßt, so befinden sich am rechten und unteren Rand Patches geringerer Größe. Die Anzahl der Patches in x- bzw. y-Richtung ergibt sich demnach wie folgt:

$$NumXPatches = \left\lfloor \frac{PicWidth}{PatchWidth} \right\rfloor + 1 \qquad (2.1a)$$

$$NumYPatches = \left\lfloor \frac{PicHeight}{PatchHeight} \right\rfloor + 1 \qquad (2.1b)$$

[3]**Beispiel:** Man stelle sich ein System aus einer differentiellen Übertragungstechnik, der lokalen Bilddatenverwaltung und einer Anzeigentechnik (wie sie z.B. im Kapitel 3 vorgestellt wird) vor. Zu Beginn liegen lokal keine Bilddaten vor, so dass die von der Anzeigetechnik angeforderten Bilddaten vollständig übertragen werden müssen. Möchte der Nutzer sich nun eine um nur wenige Pixel verschobene Ansicht des Gesamtbildes machen, so werden nicht alle zur Darstellung nötigen Bildausschnitte übertragen, sondern nur die kleinen, lokal noch nicht vorhandenen Teile. Dadurch entsteht eine hohe Fragmentierung des Gesamtbildes.

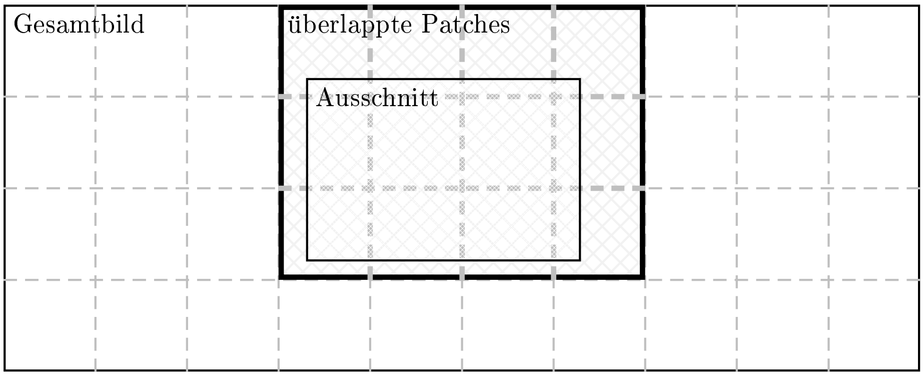


Abbildung 2.1: Aufteilung des Gesamtbildes in Patches. Nur die vom übertragenen Bildausschnitt überlappten Patches müssen lokal physisch angelegt werden. Die anderen Patches werden virtuell verwaltet.

mit

$$\begin{aligned} NumXPatches, NumYPatches &\Rightarrow \text{Patches in } x\text{- bzw. } y\text{-Richtung} \\ PicWidth, PicHeight &\Rightarrow \text{Breite, Höhe des Gesamtbildes} \\ PatchWidth, PatchHeight &\Rightarrow \text{Breite, Höhe eines Patches} \end{aligned}$$

Die Patchgröße ist bei der Instanzierung der lokalen Bilddatenverwaltung sinnvoll zu wählen. Ist sie zu groß, so reduziert sich der Effekt der Platzersparnis, da dann die Region der durch einen Bildausschnitt überlappten Patches im Durchschnittsfall größer ist. Wählt man die Patchgröße zu klein, so erhöht sich der Overhead durch die Verwaltung, was sich vor allem negativ auf die Antwortzeiten zur Bereitstellung der Bilddaten auswirkt.

2.3.2 Konzept der Auflösungsdateien

Durch die LoD–Unterstützung der Übertragungstechnik werden die Bilddaten in bestimmten Detailstufen übertragen. Die lokale Bilddatenverwaltung unterstützt in diesem Zusammenhang nur die Verwaltung verschiedener Auflösungsstufen, jedoch nicht andere Levels of Detail–Konzepte. Es ist leicht ersichtlich, dass es sinnvoll ist, für jede dieser Detailstufen eine Auflösungsdatei[4] zu generieren, in der die Bilddaten lokal abgelegt werden.[5]

Im Zusammenhang mit dem „Konzept des Tiling“ muss für jede Auflösungsstufe eine sinnvolle Patchgröße gefunden werden. Es bieten sich zwei Alternativen an:

1. **Auflösungsabhängige Unterteilung.** Die Patchgröße wird für jede Detailstufe so definiert, dass zu einem bestimmten Patch einer Auflösungsstufe *ein* Patch einer anderen Auflösung gefunden werden kann. Beide

[4] Auch Resolutionfile genannt.

[5] Natürlich wäre es auch möglich, alle Detailstufen innerhalb einer Datei zu verwalten. Jedoch würde dies den Aufbau der Auflösungsdatei wesentlich komplizieren und eine separate Verwendung einzelner Auflösungen erschweren.

Patches sollen also die gleiche Bildinformation enthalten, jedoch in einer unterschiedlichen Detailstufe.

Die jeweilige Patchgröße der Auflösungsstufe i berechnet sich demnach aus der vorzugebenden Patchgröße der 1:1-Auflösung wie folgt:

$$PatchWidth_i = \frac{PatchWidth_1}{XRes_i} \tag{2.2a}$$

$$PatchHeight_i = \frac{PatchHeight_1}{YRes_i} \tag{2.2b}$$

mit

$XRes_i, YRes_i \Rightarrow$ Auflösungszahl in x- bzw. y-Richtung der i.ten Auflösungsstufe

Probleme treten dann auf, wenn die Patchgröße der 1:1-Auflösung nicht ganzzahlig durch eine Auflösungszahl teilbar ist. Die notwendige Rundung verhindert dann wieder eine eindeutige Zuordnung zwischen den Patches verschiedener Auflösungen. Zudem werden die Patchgrößen für hohe Auflösungszahlen sehr klein, was den Verwaltungsaufwand erhöht.

2. **Auflösungsunabhängige Unterteilung.** Jeder Auflösungsstufe wird die gleiche Patchgröße zugewiesen. Somit ist es nicht möglich, zu einem Patch einer Auflösung *ein* Patch einer anderen Auflösung zu finden. Der Vorteil dieses Ansatzes besteht darin, dass für jede Auflösungsstufe eine sinnvolle, ganzzahlige Patchgröße existiert, ohne den Verwaltungsaufwand bei höheren Auflösungszahlen zu erhöhen.

In der lokalen Bilddatenverwaltung wurde der zweite Ansatz aufgrund der genannten Vorteile umgesetzt.

2.3.3 Konzept des Aufbaus der Auflösungsdateien

Dieser Abschnitt liefert eine grobe Beschreibung der Struktur der Auflösungsdateien und fokussiert auf die zugrundeliegenden Konzepte. Eine exakte implementierungstechnische Dokumentation findet sich im Anhang A.

Eine Auflösungsdatei der lokalen Bilddatenverwaltung besteht aus den folgenden drei Komponenten:

- Metabildinformation bzw. Header
- Offset Tabelle
- Bilddaten

Der Abschnitt der Metabildinformation enthält Informationen zur Größe des Gesamtbildes, der Patchgröße, der Auflösung sowie der Farbtiefe der Bilddaten.

Die Offset Tabelle setzt im Zusammenhang mit dem Abschnitt der Bilddaten das „Konzept des Tiling“ um. Sie enthält für jedes Patch einen Verweis auf den Beginn seiner Bilddaten. Dazu werden die Patches im Gesamtbild von

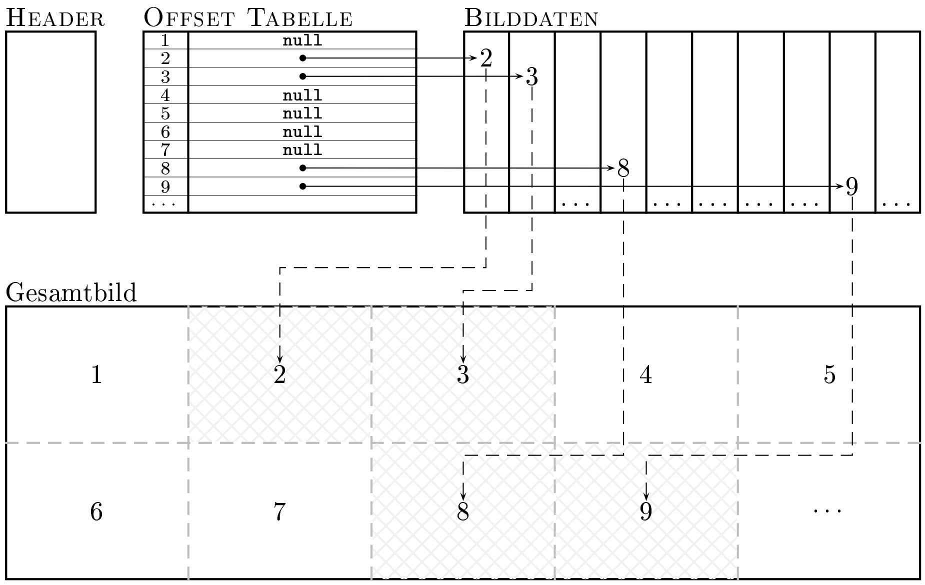


Abbildung 2.2: Grundlegender Aufbau der Auflösungsdateien. Die mit Kreuzmuster unterlegten Patches im Gesamtbild sind die lokal vorhandenen Bildteile. In der OFFSET TABELLE findet sich an der entsprechenden Stelle ein Verweis auf die BILDDATEN des jeweiligen Patches. Die Anordnung der Patches innerhalb der BILDDATEN–Sektion ist beliebig. Lokal nicht vorhandene Patches werden anhand des Eintrages `null` in der OFFSET TABELLE identifiziert.

oben links nach unten rechts durchnummeriert. Entsprechend dieser Ordnung werden die Verweise in der OFFSET TABELLE sortiert.[6] Entspricht ein Verweis dem Wert `null`, so befinden sich seine Bilddaten noch nicht in der lokalen Bilddatenverwaltung. Die Position Pos (innerhalb der OFFSET TABELLE) des Verweises auf das Patch in Zeile Z und Spalte S läßt sich wie folgt berechnen (Nummerierung beginnt mit 0; ein Verweis hat die Länge ℓ):

$$Pos = [\text{Beginn der OFFSET TABELLE}] + \ell \cdot (S + Z \cdot [\text{Anzahl der Spalten}])$$

Im Abschnitt der BILDDATEN findet man die Bildinformationen der einzelnen Patches entsprechend den Verweisen innerhalb der OFFSET TABELLE. Die Reihenfolge der Übertragung und Dekodierung bestimmt die Position der Patches innerhalb dieses Abschnitts, da diese in dieser Reihenfolge an das Ende der Auflösungsdatei angehängt werden.

Abbildung 2.2 zeigt den grundlegenden Aufbau der Auflösungsdateien.

2.3.4 Betrachtungen zur Redundanz

Bei der Instanzierung einer lokalen Bilddatenverwaltung wird es dem Programmierer freigestellt zu entscheiden, welche Auflösungen verwaltet werden sollen

[6] Am Beginn der OFFSET TABELLE findet sich also der Verweis auf die Bilddaten des linksoberen Patches und am Ende der Tabelle steht der Verweis auf die Bilddaten des rechtsunteren Patches.

und welche nicht. Das Speichern von Auflösungen neben der 1:1–Auflösung ist immer mit einer Redundanz verbunden, denn Bilddaten niedrigerer Auflösungen können ohne mathematische Probleme aus höher aufgelösten Bilddaten generiert werden. Ist die Redundanz sehr hoch, so werden die Speicherressourcen des lokalen Rechners stark belastet, aber die zur Generierung von Bilddaten niedrigerer Auflösungen benötigte Rechenzeit sinkt. Dies erhöht die Geschwindigkeit der Anzeigetechnik. Wählt man andererseits eine geringe Redundanz, so wird mehr Rechenzeit zur Generierung von Auflösungen benötigt, jedoch sinkt somit auch der Speicherplatzbedarf.

Es ist also ein Mittelweg zu finden, der sowohl Rechenzeit und Speicheraufwand in einem gewissen Rahmen hält. In diesem Abschnitt soll die Redundanz vor allem in Bezug auf den Speicheraufwand untersucht werden. Eine Abschätzung der Rechenzeit zur Generierung von Auflösungen ist nur im engen Zusammenhang mit der Anzeigetechnik möglich und wird an dieser Stelle nicht untersucht.

Zur Abschätzung der Redundanz ist es notwendig, die maximal mögliche Redundanz zu ermitteln, die entsteht, wenn man eine (sicher unrealistische) unendliche Anzahl von Auflösungen annimmt.

Im folgenden soll die Redundanz berechnet werden, die entsteht, wenn man nur 2er-potenzpixelquadratische Auflösungen speichert und damit der Beweis erbracht werden, dass die Menge der dazu notwendigen Daten konvergiert.[7]

$$\lim_{n\to\infty}\sum_{i=0}^{n}\frac{1}{4^i} \tag{2.3}$$

Setzt man in (2.3) $n=\infty$, so erhält man eine unendliche geometrische Reihe der Form $s=\sum_{i=0}^{\infty}a_1q^i$ mit $a_1=1$ und $q=\frac{1}{4}$. Die Summe der Reihe berechnet sich somit nach $s=\frac{a_1}{1-q}$. Es ergibt sich

$$\lim_{n\to\infty}\sum_{i=0}^{n}\frac{1}{4^i}=\frac{4}{3} \tag{2.4}$$

Somit wird unter der Voraussetzung, dass nur 2er-potenzpixelquadratische Auflösungen in der lokalen Bilddatenverwaltung gespeichert werden, niemals mehr als ein Drittel Redundanz erzeugt.

Ein Beispiel aus der Anwendung mit dem FishEye–View (vorgestellt in Kapitel 3) soll zeigen, dass die Redundanz und damit auch die Menge der Daten divergieren kann. Die durch den FishEye–View benötigten Auflösungen werden in Abbildung 2.3 anhand eines Beispiels aufgezeigt. Würde man alle Auflösungen speichern, die zum Aufbau des FishEye–Views notwendig sind,

[7] Unter der Voraussetzung, dass die Menge der Bilddaten in der 1:1–Auflösung endlich ist.

1:1			
1:2	2:1		
1:4	4:1	2:2	
2:4	4:2	1:8	8:1
2:8	8:2	4:4	
4:8	8:4		
8:8			

Abbildung 2.3: Benötigte Auflösungen für einen FishEye–View mit 3 Ringen. (Beispiel)

käme man auf folgende Berechnung der Menge der Daten.

$$\begin{aligned}
\lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{1}{4^i}+\sum_{j=0}^{i-1}\frac{2}{2^{i+j}}\right) &= \\
\lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{1}{4^i}+\frac{1}{2^{i-1}}\sum_{j=0}^{i-1}\frac{1}{2^j}\right) &= \\
\lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{1}{4^i}+\frac{2^i-1}{2^{i-1}}\right) &= \\
\lim_{n\to\infty}\sum_{i=0}^{n}\left(\frac{1}{4^i}+\frac{(2^i-1)2^{i+1}}{2^{2i}}\right) &= \\
\lim_{n\to\infty}\sum_{i=0}^{n}\frac{1+(2^i-1)2^{i+1}}{4^i} &= \\
\lim_{n\to\infty}\sum_{i=0}^{n}\frac{1+2\cdot 4^i-2\cdot 2^i}{4^i} &= \infty
\end{aligned} \tag{2.5}$$

Dieses Beispiel zeigt also, dass man bei der Auswahl der in der lokalen Bilddatenverwaltung zu speichernden Auflösungen darauf achten muss, nicht zuviel Redundanz zu erzeugen, da sonst der Speicherplatzbedarf enorm ansteigen kann. Es ist sicherlich sinnvoll, eine Auswahl von Auflösungen zu benutzen, deren erzeugte Redundanz zumindest konvergiert (wie es z.B. bei der Menge der 2er-potenzpixelquadratischen Auflösungen der Fall ist).

2.3.5 Konzept des Fallback

Dieses Konzept implementiert die Forderung nach einer „Interpolation von Bilddaten höherer LoD aus niedrigeren LoD“. Finden sich in der angeforderten Auflösungsstufe keine Bilddaten für ein bestimmtes Patch, so wird versucht, diese Bilddaten aus niedrigeren Auflösungen zu laden. Da sich zu einem Patch in der einen Auflösungsstufe kein einzelnes Patch in einer anderen Detailstufe zuordnen läßt, wird eine neue Anfrage an die lokale Bilddatenverwaltung gestellt, die einen Bildausschnitt mit den Koordinaten des fehlenden Patches in geringerer Auflösung anfordert.

Die Skalierung der Bilddaten auf die benötigte (höhere) Auflösung erfolgt mittels eines auf Geschwindigkeit optimierten Algorithmus aus [Möl92].

2.4 Zugriffsoptimierung

Durch das „Konzept des Tiling“ werden die Bilddaten in einzelnen Patches in den Auflösungsdateien gespeichert. Fordert die Anzeigetechnik nun einen gewissen Bildausschnitt an, so müssen die für diesen Bildausschnitt erforderlichen Patches bestimmt und einzeln geladen werden. Der angeforderte Bildausschnitt wird also stückweise konstruiert. Dieser Prozess sollte bezüglich seiner Geschwindigkeit optimiert werden. Dazu werden im Abschnitt 2.4.1 zwei Methoden miteinander verglichen.

Es ist davon auszugehen, dass die Anzeigetechnik in einer bestimmten Zeitspanne bestimmte Bildteile wiederholt anfordert. Insbesondere während der interaktiven Manipulation der Ansicht durch den Nutzer tritt ein solches Verhalten auf. Eine drastische Verringerung der Antwortzeiten ist hierbei zu erreichen, indem die häufig angeforderten Bildteile im Hauptspeicher gehalten werden. Somit entfällt die aufwendige Konstruktion der Bildteile aus den Patches zumindest teilweise. Dieser Ansatz wird durch den Speichercache der lokalen Bilddatenverwaltung umgesetzt, dessen Funktionsweise im Abschnitt 2.4.2 näher erläutert wird.

2.4.1 Optimierung des direkten Festplattenzugriffs

Ein durch die Anzeigetechnik angeforderter Bildausschnitt muss zumeist aus mehreren Patches zusammengesetzt werden. Dazu wird in jedem Fall zunächst der Speicherbereich allokiert, in den die Bilddaten des angeforderten Ausschnitts kopiert werden sollen. Nun müssen die Daten jedes beteiligten Patches einzeln aus der angeforderten Auflösungsdatei gelesen und an die entsprechende Stelle im Bildausschnitt kopiert werden. Für diesen Vorgang gibt es zwei Möglichkeiten.

1. **FSEEK–Methode.** Die Daten werden zeilenweise aus dem Patch in der Auflösungsdatei gelesen. Dies geschieht ohne Zwischenpuffer, so dass die entsprechende Pixelzeile direkt an ihre Position im Bildausschnitt eingelesen wird. Überlappt der Ausschnitt nicht vollständig die Pixelzeile, so genügt es, nur den überlappten Teil der Zeile zu laden.[8]

2. **MemCpy–Methode.** Die Daten werden patchweise aus der Auflösungsdatei in einen Zwischenpuffer eingelesen. Von dort aus werden sie zeilenweise in den Speicherbereich des angeforderten Bildausschnitts kopiert.[9]

Der Vorteil der FSEEK–Methode besteht darin, dass nur soviele Daten aus der Auflösungsdatei geladen werden wie benötigt. Andererseits muss somit für jede Zeile ein Festplattenzugriff erfolgen. Bei der MemCpy–Methode wird für

[8] In diesem Fall muss der Dateizeiger manuell auf den Beginn der nächsten Zeile gesetzt werden. Dies geschieht durch eine sogenannte `FSEEK`–Operation. Daher stammt der Name für diese Methode.

[9] Dies geschieht mittels einer `MemCpy`–Operation. Daher stammt der Name für diese Methode.

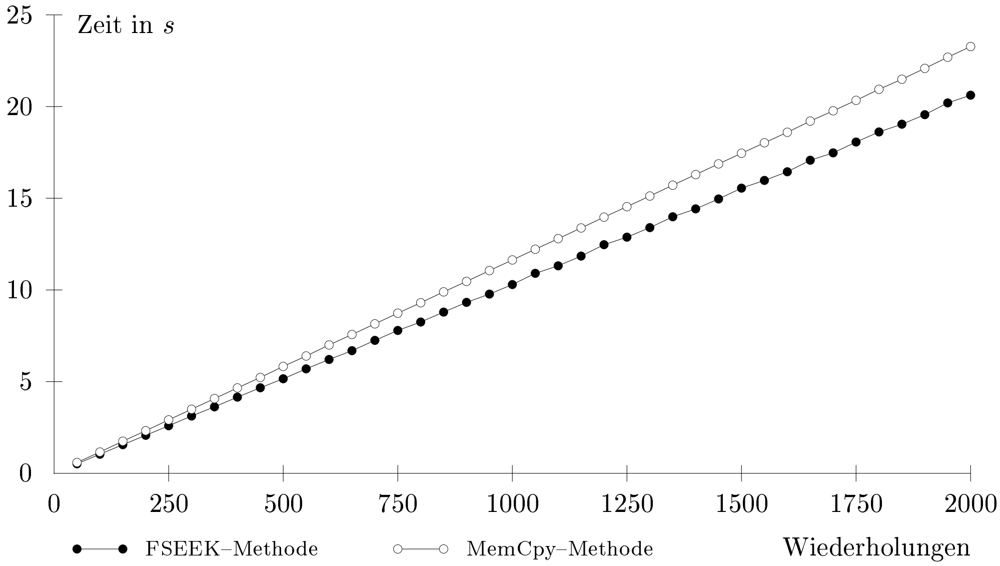


Abbildung 2.4: FSEEK– und MemCpy–Methode im Vergleich in Bezug auf die Geschwindigkeit. Eine Wiederholung beinhaltet das simulierte Laden von 100 Patches der Größe 64×64 Pixel.
Testsystem: Pentium II 400MHz, 128MB Hauptspeicher, 100MHz Frontside, U2W-SCSI Adapter, U-SCSI Festplatte (7200U/min, 9.5ms mittlere Zugriffszeit, 500KB Cache, NTFS), Windows NT 4.0
Testdaten: 1.4MB für jede Methode. Größere Testdaten erbringen nahezu gleiche Ergebnisse.

jedes Patch nur einmal auf die Festplatte zugegriffen. Andererseits werden mehr Daten geladen als benötigt. Zudem muss ein Zwischenpuffer in der Größe eines Patches allokiert werden.

Eine theoretische Abschätzung der Geschwindigkeiten der beiden Methoden ist kaum möglich, da die (unbekannten) Interna des Betriebsystems und der Hardware beim Festplattenzugriff eine entscheidende Rolle spielen. Es bietet sich somit eine praktische Abschätzung durch ein Testprogramm an, welches beide Algorithmen implementiert und die Zeit für ein wiederholtes Ausführen eines jeden Algorithmus misst. Der Quelltext des Testprogramms findet sich im Anhang B.

Abbildung 2.4 zeigt die Auswertung eines Testlaufs mit den in der Bildunterschrift angegebenen Parametern. Die FSEEK–Methode ist demnach ca. 10% schneller als die MemCpy–Methode. Wie nicht anders zu erwarten war, verlaufen die Messwerte nahezu linear, da es sich nur um ein wiederholtes Ausführen eines Algorithmus mit nahezu konstantem Zeitaufwand handelt.

Besonders interessant ist die Tatsache, dass nahezu gleiche Ergebnisse mit wesentlich größeren Testdaten[10] erzielt wurden. Ob dies bei einem weniger leistungsfähigen System ähnlich ist, konnte nicht überprüft werden.

[10] Es wurden auch Tests mit Testdaten von 24MB durchgeführt.

Jedoch wurde ein Versuch unternommen, bei dem die Daten auf einer Diskette gespeichert waren. Es sollte herausgefunden werden, ob sich die mehrfachen Datenträgerzugriffe der FSEEK–Methode negativ bei langsamen Datenträgern auswirken. Dies ist bei einer geringen Anzahl von Wiederholungen eines simulierten Ladevorgangs der Fall. Für bis zu 200 Wiederholungen ist die MemCpy–Methode schneller. Sind jedoch die Daten der Diskette erstmal vollständig im Cache des Betriebsystems,[11] erreicht die FSEEK–Methode bessere Werte. Es ist zu beachten, dass der Test mit der Diskette zwar grundlegende Vorteile bzw. Probleme der beiden Methoden aufzeigt, aber für die heutigen, hochperformanten[12] Systeme keine Relevanz besitzt.

In der lokalen Bilddatenverwaltung wurde die FSEEK–Methode aufgrund ihrer höheren Geschwindigkeit umgesetzt.

2.4.2 Optimierung durch Speichercache

Wie bereits erläutert, kann eine deutliche Verringerung der Antwortzeiten auf Anforderungen seitens der Anzeigetechnik erzielt werden, indem häufig angeforderte Bildteile im Hauptspeicher verwaltet werden. Die Bilddaten werden zunächst in diesem Speichercache gesucht und nur die Teile, die dort nicht gefunden werden konnten, werden mittels der FSEEK–Methode aus den Auflösungsdateien geladen.

Ein erfolgreiches Finden der gesuchten Daten im Speichercache wird *Hit* genannt. Es werden zwei Arten von Hits unterschieden:

1. **PicHit.** Der gesuchte Bildausschnitt existiert vollständig im Speichercache und kann ohne weiteres übernommen werden. Hierbei ist ein Kopieren der Daten nicht notwendig, lediglich die Zeiger auf die Speicherbereiche müssen angepasst werden. Im Falle eines PicHits wird die Antwortzeit der lokalen Bilddatenverwaltung minimal.

2. **PatchHit.** Der gesuchte Bildausschnitt existiert teilweise im Speichercache und die vorhandenen Teile können kopiert werden. Wenn verschiedene im Cache vorhandene Bildausschnitte den gesuchten Bildausschnitt überlappen, werden zunächst die Daten mit der größten Überlappung in x-Richtung zeilenweise kopiert, denn dies sind die „wertvollsten“ Daten im Speichercache in Bezug auf den gesuchten Ausschnitt, weil sie innerhalb einer Zeile möglichst viele Patches überlappen. Findet man Daten im Cache, die sich mit dem gesuchten Ausschnitt nur in der Breite eines Patches oder weniger überlappen, so ist der Vorteil des Speichercache gegenüber der FSEEK–Methode sehr gering, da auch in diesem Fall diese Patches einzeln in den angeforderten Ausschnitt übertragen werden müssen. Somit kommt den Daten mit der größten Ausdehnung in x-Richtung die größte Bedeutung zu.

Mit einer einfachen Methode läßt sich die Anzahl der PicHits wesentlich erhöhen. Dehnt man die Koordinaten aller angeforderten Bildausschnitte auf

[11] Während des Versuchs wurde streckenweise nicht mehr physisch auf die Diskette zugegriffen, da die Daten durch das Betriebssystem gecacht wurden.

[12] Im Vergleich zu einer Diskette.

die Grenze der überlappten Patches,[13] erhöht sich die Chance, im Speichercache einen Bildausschnitt mit den gleichen Koordinaten zu finden. Natürlich sind somit zunächst mehr Daten aus der jeweiligen Auflösungsdatei zu laden, aber diese Daten sind für alle angeforderten Bildausschnitte relevant, die in ihrer ursprünglichen Ausdehnung die entsprechenden Patches überlappen.

Die Verwaltung der Bildausschnitte im Speichercache erfolgt mittels einer sortierten Liste für alle Auflösungen. Besonders wichtig für die Verwaltung des Cache ist eine Opferstrategie, anhand derer wenig benutzte Elemente zugunsten neuer Bildausschnitte entfernt werden. Für jedes Element im Cache wird ein LRU–Zähler[14] geführt, der immer dann erhöht wird, wenn dieses Element *keine* Bilddaten zur Beantwortung einer Anforderung bereitgestellt hat. Wurden jedoch Daten aus diesem Element entnommen, um damit den angeforderten Bildausschnitt zumindest teilweise zu füllen, so wird der LRU–Zähler auf den Wert 0 gesetzt.

Die Opferstrategie des Speichercache basiert hauptsächlich auf der Auswertung des LRU–Zählers. Der Nutzer kann verschiedene Kriterien angeben, die der Cache erfüllen muss. Falls er sie nicht erfüllt, werden Elemente aus dem Cache entfernt bzw. hinzugefügt. Die Kriterien werden in einer Liste verwaltet. Höchste Priorität kommt somit dem ersten Element der Kriterienliste zu. Folgende Kriterien stehen zur Verfügung:

1. **Maximale Anzahl von Elementen.** Wird die maximal erlaubte Anzahl von Elementen im Speichercache überschritten, so muss jeweils das Element mit dem höchsten LRU–Zähler entfernt werden bis die maximale Anzahl wieder erreicht ist.

2. **Minimale Anzahl von Elementen.** Wird die minimal erlaubte Anzahl von Elementen im Cache unterschritten, so wird das aktuell hinzuzufügende Element ohne die Betrachtung weiterer Kriterien hinzugefügt. Kriterien, die in der Kriterienliste hinter diesem Minimal–Kriterium stehen, werden bei dessen Erfüllung also nicht mehr überprüft.

3. **Maximale Speichernutzung.** Belegen *alle* Elemente im Cache zusammen mehr Speicher als maximal erlaubt, so werden nacheinander die Elemente mit dem jeweils höchsten LRU–Zähler entfernt bis dieses Kriterium erfüllt wird.

4. **Maximaler LRU–Zähler.** Alle Elemente, deren LRU–Zähler einen höheren Wert als maximal erlaubt aufweisen, werden entfernt.

5. **Identische UserID.** Der programmiertechnische Benutzer des Speichercache kann dem angeforderten Bildausschnitt eine Kennzeichnung (UserID) in Form einer ganzen Zahl geben. Der angeforderte Bildausschnitt verdrängt somit bei seiner Einlagerung im Speichercache alle Elemente mit der gleichen UserID. Dies ist für Anzeigetechniken relevant, deren angeforderte Bildausschnitte sich einer *logischen* Position in der Darstellung zuordnen lassen. Ein Beispiel hierfür ist der in Kapitel 3 vorgestellte Fish-Eye–View. Bei dieser Anwendung ist es nicht sinnvoll, zwei dem Fokus zuzuordnende Bildausschnitte im Speichercache einzulagern. Mittels einer

[13] Vergleiche Abbildung 2.1 auf Seite 15.
[14] **L**east **R**ecently **U**sed

UserID lässt sicht dies verhindern, da der jeweils vorherige Bildausschnitt mit der gleichen UserID verdrängt wird.

6. **Leeres Kriterium.** Dieses Pseudo–Kriterium dient zur Kennzeichnung des Endes der Kriterienliste.

Diese Kriterien können innerhalb der Kriterienliste beliebig angeordnet werden, jedoch jeweils nur einmal vorkommen. Diese Liste (und somit die Opferstrategie des Speichercache) ist auch während der Ausführung des Programms anpassbar.

Im Zusammenhang mit der Anzeigetechnik des FishEye–Views[15] ergibt die Anwendung des Speichercache eine drastische Reduzierung der Antwortzeiten. Die Statistik des Speichercache weist PicHits für über 50% der angeforderten Bildausschnitte auf.[16] PatchHits kommen wesentlich seltener vor. Nur etwa 10% bis 20% der angeforderten und nicht durch PicHits beantworteten Patches können aus dem Speichercache mit Daten gefüllt werden. Eventuell ist eine Erhöhung dieses Prozentsatzes möglich, wenn kleinere Patchgrößen benutzt werden. Dies wiederum verringert díe Wahrscheinlichkeit eines PicHits.

[15] Vorgestellt in Kapitel 3

[16] Unter der Vorraussetzung, dass die Koordinaten des Ausschnitts auf die Grenze der überlappten Patches ausgedehnt werden. Vergleiche dazu Abbildung 2.1 auf Seite 15. Ohne diese Koordinatenänderung sind es je nach Interaktionsgeschwindigkeit zwischen 1% und 15%.

Kapitel 3

Der rechteckige FishEye–View

3.1 Motivation und Stand der Forschung

Der Visualisierung von Information kommt eine große Bedeutung zu, da sich Zusammenhänge in Bildern sehr schnell erschließen lassen. Es gibt viele Algorithmen, Information sinnvoll und leicht verständlich darzustellen - sei es mit spezieller Vorverarbeitung (z.B. automatische Klassifizierung) oder ohne. Unabhängig davon, welche Art der Visualisierung man wählt, ist jedoch zumeist so viel darzustellen, dass ein normaler Bildschirm dafür nicht ausreicht, und selbst wenn man eine besonders große Darstellungsfläche wählen würde, hätte man immer noch das Problem, diese Menge von Informationen mit all ihren Details zu begreifen. Das ist der Punkt, an dem die in diesem Kapitel vorgestellte Technik ansetzen soll: Ausgangspunkt ist visuelle Information (z.B. eine Grafik), und es wird eine platzsparende, aber dennoch detaillierte Darstellung gesucht.

Es gibt eine Reihe von Techniken, die dieses elementare Problem zu lösen versuchen. Sarkar und Brown unterscheiden in [SB92] drei wesentliche Ansätze:

1. Darstellung der Information in einer Ansicht.

2. Darstellung von zwei oder mehreren Ansichten - eine für die undetaillierte Gesamtübersicht und die andere für einen vergrößerten, detaillierten Ausschnitt.

3. Integration von Fokus und Kontext in einer Ansicht.

Die erste Methode hat das Problem, dass die komplette Ansicht entweder nicht auf den Bildschirm passt oder aber die Details nur sehr klein oder gar nicht dargestellt werden können. Einen Vorteil bietet dabei schon die zweite Technik: Ein bestimmter Ausschnitt wird groß genug dargestellt, so dass auch alle Details ausreichend erkennbar sind. Eine zweite Ansicht zeigt, welchen Teil der Gesamtansicht der Ausschnitt zeigt. Vorteile sind ganz klar der hohe Detailgrad und die bessere Navigation gegenüber der ersten Technik. Sarkar und Brown sehen die Nachteile darin, dass mehrere Ansichten auch mehr Platz auf dem

Bildschirm beanspruchen und der Nutzer gezwungen ist, die verschiedenen Ansichten mental zu kombinieren.

Der dritte Ansatz beruht darauf, in *einer* Ansicht wichtige Bereiche der Darstellung detailliert und groß zu zeigen (*zu fokussieren*) und die umrandenden Gebiete kleiner und weniger detailliert darzustellen (*zu kontextieren*). Es handelt sich um sogenannte Fokus-und-Kontext-Techniken, die einen hohen Detailgrad und gute Navigation ermöglichen, ohne die Nachteile der ersten beiden Methoden zu erben. Jedoch kommt es durch die Integration von Fokus und Kontext in einer Ansicht bei diesen Techniken zu Verzerrungen gegenüber den ersten beiden Ansätzen. Wie stark diese Verzerrungen sind, ist im Einzelfall zu untersuchen.

Eine Fokus-und-Kontext-Technik ist in ihrer Ursprungsform bereits seit langem bekannt: In der Photographie erzeugt man ein Bild mit einem FishEye-Effekt durch eine Weitwinkeloptik, die nahstehende, fokussierte Objekte vergrößert und alle anderen Objekte zum Rand hin abfallend kleiner darstellt. Somit ist es möglich, sich das fokussierte Objekt genauer anzusehen, ohne den Überblick über das gesamte Bild zu verlieren.

Der FishEye–View wurde 1982 von George W. Furnas zum ersten Mal für Belange der rechnergestützten Visualisierung eingesetzt ([Fur82]). Furnas' Arbeit sieht jedoch nicht vor, bestimmte Ausschnitte der Grafik zu vergrößern oder zu verkleinern anhand ihres Abstandes zum Fokus, sondern Teile der Grafik (respektive Knoten eines Graphen) werden aus der Darstellung entfernt, wenn sie ein bestimmtes Abstandsmaß überschreiten. Somit geht es hierbei nicht darum, ein bereits erzeugtes Layout platzsparend zu präsentieren, sondern das Layout selbst wird verändert, um dieses Ziel zu erreichen. Der Ansatz bezieht sich auf strukturierte Daten, wie z.B. Graphen oder Quelltexte.

Manojit Sarkar und Steven P. Reiss stellen in [SR92] eine Erweiterung vor. Hier bezieht sich das Abstandsmaß auf den euklidischen Abstand innerhalb des bereits angefertigten grafischen Layouts. Sie geben eine Antwort auf die von Furnas aufgeworfene Frage nach mehreren Foki und stellen zwei verschiedene Stretchtechniken zur Verzerrung des Ausgangsbildes vor ([SSTR93]). Eine dieser Techniken (das orthogonale Stretching) ist der grundlegende Ansatz der in diesem Kapitel vorgestellten Darstellungsform. Der andere Ansatz (das polygonale Stretching) basiert auf der Warping-Technik, die in der Computer Animation beim Morphing eingesetzt wird. Diese Art der Layoutverzerrung ist wesentlich rechenaufwendiger und eignet sich (zumindest unter der Maßgabe kurzer Antwortzeiten während der Interaktion) nur für grafische Layouts. Sie bietet jedoch die größtmögliche Integration von Fokus und Kontext durch minimale Verzerrung des Gesamtlayouts. Sarkar u.a. befassen sich in ihren entsprechenden Arbeiten nicht mit der Frage nach einer abgestuften Kontextierung und der Möglichkeit des Zooms in den FishEye–View selbst.

Die in diesem Kapitel vorgestellte Technik basiert auf der Verarbeitung von Rasterbildern. Es werden beliebig viele Kontextringe mit beliebigen Größenverhältnissen zueinander unterstützt.[1] Weiterhin ist ein Zoom in den FishEye–View selbst möglich. Die Technik ist außerdem vollständig auf multiple Foki erweiterbar. Ohne weiteres sind auch grafische Layouts anwendbar, obwohl die

[1] Dies beinhaltet eine variabel abgestufte Kontextierung. Die Definition erfolgt im Abschnitt 3.3.2 ab Seite 32.

Techniken von Sarkar u.a. auf diesem Gebiet eindeutig Vorteile haben.

Die Vorteile des rechteckigen FishEye–Views werden besonders im mobilen Kontext bei der Übertragung mittels Wavelets deutlich, wie es in dieser Arbeit skizziert wird. Die Darstellungsform ermöglicht es, rechteckige Regions of Interest mit verschiedenen Levels of Detail[2] zu definieren. Damit wird sichergestellt, dass nur die Menge an Informationen übertragen wird, die auch tatsächlich dargestellt wird. Das heißt: Keine Region des Bildes wird in einem höheren Levels of Detail übertragen als mittels des FishEye–Views dargestellt.

3.2 Anforderungen

Wie bereits im Kapitel 1 beschrieben, sieht das zugrundeliegende Anwendungsszenario wie folgt aus:

Bilddaten werden in Wavelet-kodierter Form von einem Server an einen (mobilen) Client übertragen und sollen dort auf dem verhältnismäßig kleinen Display vom Anwender schnell exploriert werden können.

Es ergeben sich also folgende Problemfelder mit den jeweiligen Zielstellungen, die durch die zu verwendende Darstellungstechnik entweder erreicht oder aber zumindest unterstützt werden müssen:

- Übertragung
  - minimales Datenaufkommen
- Darstellung
  - Platzersparnis
  - Verständlichkeit
  - Anpassbarkeit
- Interaktion
  - minimale Antwortzeiten

Ausgehend von einer Fokus-und-Kontext-Technik (basierend auf den Ausführungen in 3.1) ergeben sich daraus eine Reihe von Forderungen an die Darstellungstechnik, die im folgenden aufgeführt werden. Einige dieser Forderungen stellten bereits Sarkar u.a. in [SSTR93] für Techniken auf, die die Visualisierung großer Layouts auf kleiner Darstellungsfläche ermöglichen. Sie wurden hier in z.T. abgewandelter Form übernommen.[3]

[2] Das Hauptaugenmerk bei der Definition von Levels of Detail durch den FishEye–View liegt bei der Auflösung der Bilddaten. Bitebenen und Farblevels können zwar auch eine Rolle spielen, werden jedoch nicht direkt durch das Prinzip der Darstellung unterstützt.

[3] Dies bezieht sich auf folgende Forderungen: „Exakte Definition des Fokus“, „Gleichmäßige Skalierung der Fokusregion“, „Integration von Fokus und Kontext“, „Multiple Foki“ und „Ordnungserhaltende Darstellung“.

Exakte Definition des Fokus: Damit ist der Nutzer in der Lage, dass Gebiet seines Interesses genau anzugeben. Das befördert die Aussagekraft der Darstellung. Sarkar u.a. beziehen sich in diesem Punkt nicht nur auf die Grösse und Position des Fokus, sondern auch auf seine Geometrie. Sie fordern, dass jede geschlossene, konvexe Region als Fokus definiert werden kann.

Gleichmäßige Skalierung der Fokusregion: Winkel, Parallelität, Konnektivität und Proportionalität sind wichtige Grundmerkmale eines Layouts, die durch die Darstellung nicht zerstört werden sollten. Bestimmte Informationen verlieren ihre Aussagekraft, wenn bereits eines dieser Merkmale verloren geht (z.B. Straßenkarten oder Fluchtpläne).

Beeinflussung der Kontextierung: Der Nutzer sollte in der Lage sein, die Art der Kontextierung zu bestimmen. Damit ist Kontrolle über die Verzerrung des Layouts bis zu einem gewissen Maß möglich.

Wichtung von Regionen: Es sollte die Möglichkeit bestehen, eine Wichtung zwischen Fokus und Kontext sowie zwischen verschiedenen Teilen des Kontext zu definieren. Dies ermöglicht, dass für den Nutzer wichtige und unwichtige Regionen des Layouts schärfer voneinander getrennt werden können. Die vorgenommene Wichtung läßt sich dann so umsetzen, dass das Datenaufkommen bestimmter Regionen priorisiert, reduziert usw. werden kann.

Uniforme Skalierung der Kontextregionen: Wenn ganze Regionen des zugrundeliegenden Layouts in gleicher Weise skaliert werden, reduziert sich der Aufwand für die Berechnung der Darstellung. Falls die Bilddaten für eine Region aus Bilddaten anderer Auflösungen erzeugt werden müssen, so ist dieser Prozeß für Regionen mit uniformer Skalierung wesentlich schneller als für Regionen mit nichtkonstanter Auflösungsfunktion. Ein signifikanter Vorteil während der Interaktion. Andererseits entstehen dadurch an den Regionsgrenzen Auflösungssprünge, die das Verständnis für die Darstellung erschweren.

Gleichmäßige Detailierung der Kontextregionen: Wenn ganze Regionen des zugrundeliegenden Layouts in gleicher Weise detailliert werden, können konstante Levels of Detail-Parameter für die gesamte Region gefunden werden.[4] Dies ermöglicht, dass alle anzuzeigenden Bilddaten in dem übertragenen Level of Detail auch dargestellt werden. Wären die Parameter nicht konstant, so müsste man die jeweiligen Maxima der Parameterfunktionen finden und in dieser Detailstufe die Daten übertragen. Lokal würden diese Daten dann entsprechend den Parameterfunktionen in der Detailstärke reduziert. Eine gleichmäßige Detailierung erhöht also den Nutzeffekt der Übertragung.

[4] Es sei angemerkt, dass eine „Gleichmäßige Detailierung der Kontextregionen“ zwangsläufig auch eine „Uniforme Skalierung der Kontextregionen“ mit sich bringt. Die beiden Forderungen wurden voneinander getrennt, weil sie einen ähnlichen Aspekt von unterschiedlichen Seiten betrachten. Zudem ist uniforme Skalierung auch ohne gleichmäßige Detailierung sehr wohl denkbar.

Integration von Fokus und Kontext: Je weicher Fokus und Kontext in der Darstellung miteinander kombiniert werden, desto weniger wird das ursprüngliche Layout verzerrt und desto verständlicher ist damit die Darstellung. Eine hohe Integration von Fokus und Kontext ist wünschenswert, erfordert jedoch zumeist Techniken, die nichtkonstante LEVEL OF DETAIL-Parameterfunktionen nutzen. Ein Mittelweg besteht darin, die LEVEL OF DETAIL-Parameter im Kontext *stufenweise* mit steigendem Abstand vom Fokus zu verringern.

Kontrolle des Platzaufwandes: Auf kleinen Displays verliert der Nutzer oft den Überblick über die gesamte Darstellung. Es ist wünschenswert, wenn die Parameter der Darstellung so anpassbar sind, dass die Ansicht in ihrer Größe die mentalen Fähigkeiten des Nutzers nicht übersteigt. Dies erhöht auch das Grundverständnis für das visualisierte Layout.

Multiple Foki: Ein Nutzer könnte sich unterschiedliche Regionen eines Layouts gleichzeitig betrachten wollen, z.B. um sie miteinander vergleichen zu können. Für solche Fälle sollte eine Darstellungstechnik multiple Foki unterstützen. Jedoch steigt natürlich der Platzaufwand bei mehreren Foki und die Interaktion mit der Darstellung wird komplizierter und rechenaufwendiger.

Ordnungserhaltende Darstellung: Grundlegende Eigenschaften eines Layouts sollten auch bei einer verzerrten Darstellung erhalten bleiben. Dies bezieht sich auf die Anordnung, Clusterung und Topologie von Objekten. In [SSTR93] kann man genauere Ausführungen und Verweise auf ausführliche Arbeiten zu diesem Thema finden.

Unterbrechbares Zeichnen: Für die Exploration des Layouts ist eine interaktive Veränderung von Fokus- und anderen Parametern unerläßlich. Die Interaktion wiederum sollte so schnelle visuelle Ergebnisse liefern, dass der Nutzer auf jede Parameterveränderung eine Antwort erhält. Dies erhöht das Verständnis für die Darstellungstechnik. Wenn die Darstellung während eines Interaktionsschrittes nicht vollständig neu gezeichnet werden kann, weil bereits eine nächste Anforderung durch den Nutzer vorliegt, dann sollte das Zeichnen unterbrechbar sein, um zumindest auf diesem Wege kurze Reaktionszeiten zu gewährleisten.

Zoom der Gesamtdarstellung: Es ist wünschenswert, dass der Nutzer sich über die Gesamtdarstellung einen Überblick verschaffen kann, ohne jedoch deren Struktur zu verändern. Es ist ebenso vorteilhaft, sich Details der Darstellung näher betrachten zu können. Dies integriert in die Fokus-und-Kontext-Technik einen Zooming-und-Panning-Ansatz. Das Verständnis für Darstellung und Verzerrung kann erhöht werden. Nicht zu unterschätzen ist auch der Vorteil für die Übertragung: Das Bild kann zunächst bei geringem Zoom betrachtet und oberflächlich gemustert werden. Bei Interesse können die Bilddaten in einer höheren Zoomstufe übertragen werden.

Einige der obigen Forderungen ergänzen sich, andere wiegen ihre Vor- und Nachteile gegenseitig auf. Eine Zusammenstellung und Einschätzung der Forderungen in Bezug auf die Zielstellungen bietet Tabelle 3.1.

	Zielstellungen				
	Übertragung	Darstellung			Interaktion
Forderungen	min. Datenaufkommen	Platzersparnis	Verständlichkeit	Anpassbarkeit	min. Antwortzeiten
Exakte Definition des Fokus:					
Größe	$\circ$	$\circ$	$+$	$+$	$-$
Position	$\circ$	$\circ$	$+$	$+$	$\circ$
Geometrie	$\circ$	$\circ$	$+$	$+$	$--$
Gleichm. Skalierung des Fokus	$\circ$	$\circ$	$+$	$\circ$	$\circ$
Beeinflussung der Kontextierung	$\circ$	$+$	$+$	$+$	$\circ$
Wichtung von Regionen	$++$	$+$	$+$	$+$	$\circ$
Uniforme Skalierung der Kontextregionen	$\circ$	$\circ$	$-$	$\circ$	$++$
Gleichmäßige Detailierung der Kontextregionen	$++$	$\circ$	$\circ$	$\circ$	$\circ$
Integration Fokus & Kontext	$-$	$\circ$	$++$	$\circ$	$-$
Kontrolle des Platzaufwandes	$\circ$	$++$	$+$	$+$	$\circ$
Multiple Foki	$\circ$	$-$	$+$	$+$	$-$
Ordnungserhaltende Darstellung	$\circ$	$\circ$	$+$	$\circ$	$\circ$
Unterbrechbares Zeichnen	$\circ$	$\circ$	$+$	$\circ$	$++$
Zoom der Gesamtdarstellung	$+$	$+$	$+$	$+$	$\circ$

Tabelle 3.1: Zuordnung von Forderungen zu Zielstellungen. „$+$“ bedeutet, dass die Zielstellung durch die Forderung unterstützt wird. „$-$“ bedeutet, dass die Forderung der Zielstellung entgegensteht. „$\circ$“ bedeutet, dass die Forderung keine relevante Auswirkung auf die Zielstellung hat.

Welche Forderungen durch den rechteckigen FishEye–View erfüllt werden, zeigt Tabelle 3.2. Wie im Vergleich mit Tabelle 3.1 zu erkennen ist, wurde bei der Auswahl der durch den FishEye–View umgesetzten Forderungen darauf geachtet, minimales Datenaufkommen zu erzeugen und einen hohen Interaktivitätsgrad zu gewährleisten. Darunter leidet z.T. auch die Darstellungsqualität, besonders durch die nur mäßige Integration von Fokus und Kontext.

[5] Nur stufenweise Verringerung der Detailparameter mit steigendem Abstand vom Fokus.
[6] Nicht implementiert, jedoch ohne weiteres integrierbar.

Forderungen	rechteckiger FishEye–View
Exakte Definition des Fokus:	
Größe	●
Position	●
Geometrie	○
Uniforme Skalierung des Fokus	●
Beeinflussung der Kontextierung	●
Wichtung von Regionen	●
Uniforme Skalierung der Kontextregionen	●
Gleichmäßige Detailierung der Kontextregionen	●
Integration Fokus & Kontext[5]	◑
Kontrolle des Platzaufwandes	●
Multiple Foki[6]	◑
Ordnungserhaltende Darstellung	●
Unterbrechbares Zeichnen	●
Zoom der Gesamtdarstellung	●

Tabelle 3.2: Durch den FishEye–View erfüllte Forderungen. ● bedeutet, dass die Forderung umgesetzt wurde. ○ bedeutet, dass die Forderung nicht umgesetzt wurde. ◑ bedeutet, dass die Forderung nur teilweise bzw. bedingt umgesetzt wurde (siehe hierzu die jeweiligen Fußnoten).

3.3 Konzepte und Geometrie des rechteckigen FishEye–Views

Dieser Abschnitt soll zunächst die grundlegenden Konzepte der Darstellungstechnik des FishEye–Views erklären. Hierbei geht es noch nicht um exakte Formeln wie in Abschnitt 3.4, sondern die geometrischen Möglichkeiten der Darstellung sollen erfasst werden.

3.3.1 Konzept des Ausgangsbildes und des Fokus

Grundlage des FishEye–View ist das Vorhandensein einer Bitmap, dem sogenannten Ausgangsbild. Alle im FishEye–Viewdargestellten Bilddaten sind die (wenn auch zumeist skalierten) Daten des Ausgangsbildes. Es definiert die höchste Auflösung der Bilddaten, und seine Auflösungszahl wird auf 1 gesetzt. Die Auflösungszahl ist der ganzzahlige Divisor zur Skalierung der Bilddaten. Die Auflösungskombination (Kombination der Auflösungszahlen für x- und y-Richtung) des Ausgangsbildes ist 1:1. Im FishEye–View gibt es keine Teilbilder mit Bilddaten höherer Auflösung als die des Ausgangsbildes. Geringere Auflösungen haben höhere Auflösungszahlen.

Der Fokus ist ein rechteckiger Teil des Ausgangsbildes, der im FishEye–View in der höchstmöglichen Auflösung dargestellt wird. In keinem anderen Teil des FishEye–View werden Bilddaten in der gleichen oder einer höheren Auflösung dargestellt. Damit wird unter anderem die Aufmerksamkeit des Betrachters auf den Fokus gelenkt, der den zentralen Mittelpunkt des FishEye–Viewsdarstellt. Die Koordinaten des Fokus im Ausgangsbild definieren in erster Linie die Gestalt des FishEye–View.[7]

3.3.2 Konzept der Kontextringe

Die durch den Fokus nicht erfassten Bilddaten des Ausgangsbildes werden durch eine bestimmte Anzahl sogenannter Kontextringe eingeteilt und (in Rechtecken verschiedener Auflösungskombinationen) im FishEye–View dargestellt. Jeder Kontextring wird zunächst durch seine ihm zugewiesene, grundlegende Kontextauflösung[8] charakterisiert.[9] Es kann mehr Kontextauflösungen als Ringe geben. Dem Fokus nähere Kontextringe haben eine höhere Grundauflösung als weiter entfernte Ringe.[10] Auch dies befördert die Aufmerksamkeit des Betrachters auf den Fokus. Durch die geringere Auflösung der Kontextringe verbraucht der FishEye–View weniger Platz auf der Darstellungsfläche als das Ausgangsbild selbst.

Um Bildsprünge zwischen den Ringen zu vermeiden, müssen diese in Rechtecke zerteilt werden, die verschiedene Auflösungen besitzen. Zu einem nicht geringen Teil werden hierbei auch Rechtecke erzeugt, die ungleichmäßig skaliert werden, deren Auflösungszahlen für die x- bzw. y-Richtung also unterschiedlich sind. Die Auflösung eines solchen Rechteckes wird definiert durch die Kon-

[7] Neben Ratiowerten und Kontextauflösungen.

[8] Auch Grundauflösung genannt.

[9] Ein Ring mit einer Kontextauflösung von n wird deshalb auch $n$er Ring genannt, z.B. 2er, 4er oder 8er Ring.

[10] Ringe mit gleicher Auflösung sind ausgeschlossen.

textauflösung des zugehörigen Ringes und eine eventuelle Vorgabe durch innere Ringe bzw. den Fokus. Abbildung 3.1a stellt eine Aufteilung des Ausgangsbildes in Fokus, Kontextringe und Rechtecke mit zugehöriger Auflösung dar.

Die Kontextringe im FishEye–View können mit verschiedenen Größenverhältnissen zueinander dargestellt werden. Jeder Ring besitzt einen solchen ganzzahligen Ratiowert, der seine Größe (im Vergleich zu den Ratiowerten anderer Ringe) im FishEye–View bestimmt. Abbildung 3.2a zeigt die bereits in Abbildung 3.1a vorgestellte Einteilung mit unterschiedlichen Ratiowerten.

3.3.3 Konzept des Zoom

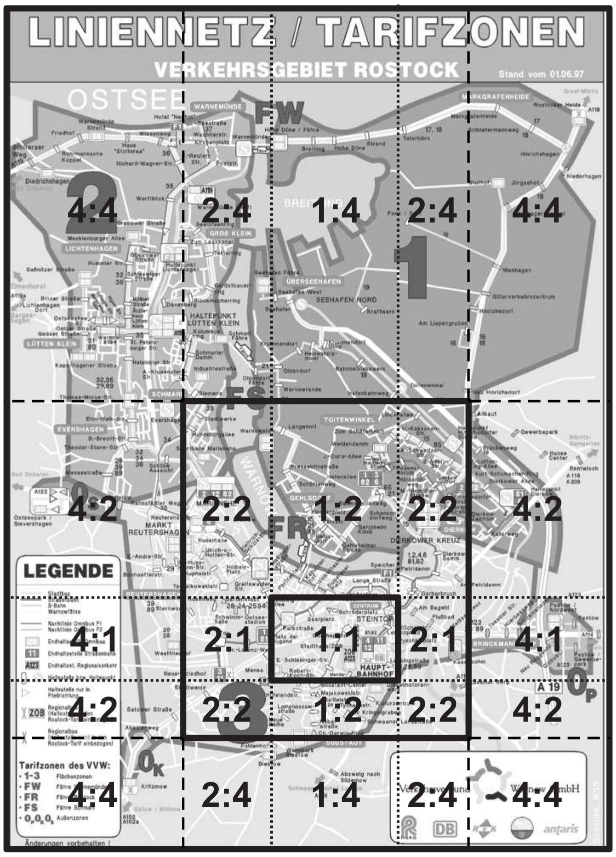
Die Gesamtdarstellung des FishEye–Views läßt sich mittels eines abgestuften Zooms verkleinern bzw. vergrößern. Das Verkleinern der Ansicht wird mit „Herauszoomen“ und das Vergrößern mit „Hineinzoomen“ bezeichnet.

Ein stufenweises Herauszoomen aus dem FishEye–View geschieht durch Verschiebung der Kontextauflösung in den nächstinneren Ring. Der Fokus erhält dabei die Auflösung des ersten Kontextringes, der erste Ring die des zweiten usw. Bleibt für den äußeren Ring keine Kontextauflösung mehr übrig,[11] so wird er nicht dargestellt und die verbleibenden Bilddaten unter den anderen Ringen aufgeteilt. Die Grenze für das Herauszoomen ist dann erreicht, wenn selbst der erste Ring keine gültige Kontextauflösung mehr besitzt. In diesem Fall werden im Fokus *alle* Bilddaten in der letzten gültigen Auflösung dargestellt.

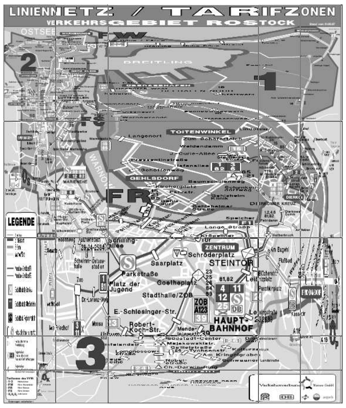
Ein Hineinzoomen in den FishEye–View geschieht in umgekehrter Weise. Die Grenze ist erreicht, wenn im Fokus die Auflösung des Ausgangsbildes, also 1:1, dargestellt wird.

Natürlich wäre es durch den Einsatz bestimmter Skalierungsalgorithmen möglich, einen stufenlosen Zoom ohne Grenzen bereitzustellen. Jedoch wurde die vorliegende Technik implementiert, um mit Wavelets kodierte und übertragene Bilddaten darzustellen. Durch die Waveletkodierung werden gewisse Auflösungen (zumeist pixelquadratische Auflösungen mit 2er-Potenz-Auflösungszahlen, also 2:2, 4:4, 8:8 usw.) vorgegeben, die somit gleich die Kontextauflösungen der Ringe definieren können. Würde man sich nun (mittels eines hypothetischen stufenlosen Zooms) zwischen zwei Stufen des vorhandenen Zooms befinden, so müßten die Bilddaten in der höheren Auflösung der beiden Zoomstufen übertragen werden. So würden sie dann allerdings nicht angezeigt, sondern mit lokal verringerter Auflösung. Das wäre ein zusätzlicher Verbrauch von Bandbreite, der zu keinem besseren visuellen Resultat führen würde. Zudem benötigte man für den Skalierungsalgorithmus eine nicht zu unterschätzende Rechenleistung, die (unter der Maßgabe großer Bilder) so manches System nicht bieten kann. Ein stufenloser Zoom ist im Kontext der (mobilen) Datenübertragung also nicht sonderlich sinnvoll, wäre jedoch wünschenswert für den Fall, dass bereits alle Bilddaten lokal vorliegen und genügend Rechenleistung vorhanden ist. Für diesen Fall wiederum sollte man für die Darstellung des FishEye–Views dann auch nicht orthogonales Stretching einsetzen, wie es Grundlage der hier vorgestellten Technik ist, sondern eine Abwandlung des in [SSTR93] vorgestellten polygonalen Stretchings.

[11] Es sei daran erinnert, dass es mehr Kontextauflösungen als Kontextringe geben kann.

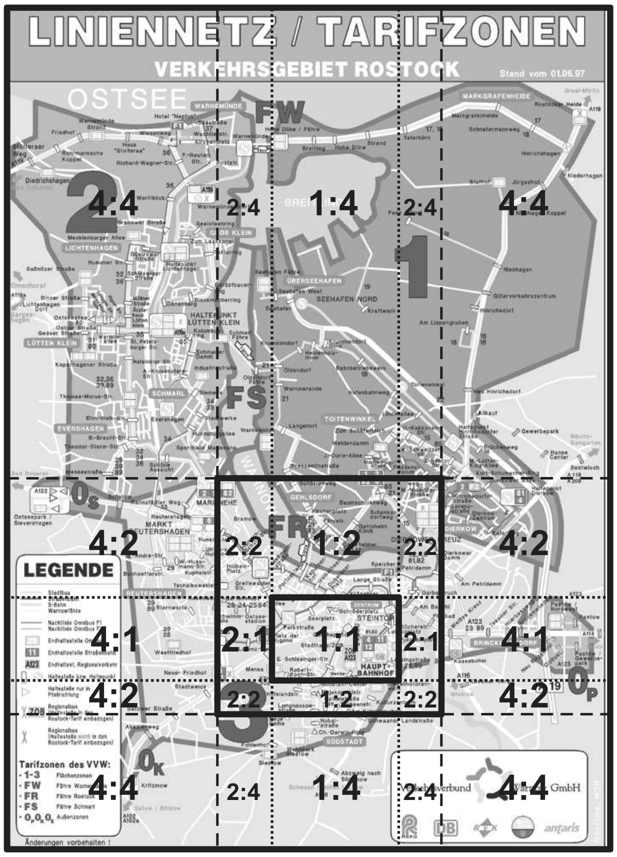


(a) Kontextauflösungen 2:2 und 4:4. Ratios 1:1:1.

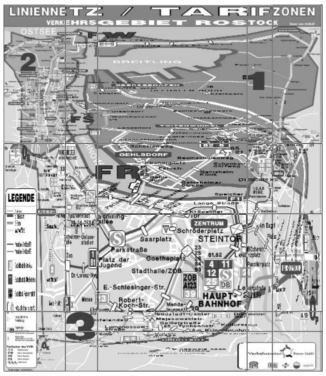


(b) Zugehöriger FishEye–View.

Abbildung 3.1: Die Geometrie des rechteckigen FishEye–Views I.



(a) Kontextauflösungen 2:2 und 4:4. Ratios 1:1:2.



(b) Zugehöriger FISHEYE–VIEW.

Abbildung 3.2: Die Geometrie des rechteckigen FISHEYE–VIEWS II.

3.4 Die Mathematik des monofokalen rechteckigen FishEye–Views

In diesem Abschnitt wird die Berechnungsvorschrift für den monofokalen rechteckigen FishEye–View vorgestellt. Es werden beliebige Anzahl der Ringe und beliebiges Größenverhältnis der Ringe zueinander unterstützt. Einzelne Ringe können an- bzw. ausgeschaltet werden. Ein Zoom ist in Stufen möglich.

Gegebene Größen:

$$\begin{aligned}
vPicWidth &\Rightarrow \text{Weite des Ausgangsbildes}\\
vPicHeight &\Rightarrow \text{Höhe des Ausgangsbildes}\\
Left1 &\Rightarrow \text{linke Koordinate des Fokus}\\
Top1 &\Rightarrow \text{obere Koordinate des Fokus}\\
Right1 &\Rightarrow \text{rechte Koordinate des Fokus}\\
Bottom1 &\Rightarrow \text{untere Koordinate des Fokus}\\
NumOfBelts &\Rightarrow \text{Anzahl der Ringe inkl. Fokus}\\
BeltRes_i &\Rightarrow \text{Auflösung des Ringes } i,\ i = 0, \ldots, NumOfBelts - 1\\
BeltRatio_i &\Rightarrow \text{Größenverhältnis-Wert[12] des Ringes } i, \text{ dito}\\
BeltVis_i &\Rightarrow \text{Sichtbarkeit des Ringes } i, \text{ dito}\\
ZoomIndex &\Rightarrow \text{Zoom, } 0 \le ZoomIndex \le NumOfBelts - 1
\end{aligned}$$

Die Ringe im FishEye–View sollen von innen nach außen gezählt werden. Beim Index $i = 0$ handelt es sich also um die Fokusregion, während $i = NumOfBelts - 1$ den äußersten Ring bezeichnet (siehe Abbildung 3.3 auf Seite 39).
Es sei

$$BeltRes_j < BeltRes_i \qquad \text{für alle } j < i \tag{3.1}$$

und

$$BeltVis_i = \begin{cases} 0 & \text{Ring } i \text{ ist unsichtbar} \\ 1 & \text{Ring } i \text{ ist sichtbar} \end{cases}$$

und

$$BeltVis_0 = 1 \qquad BeltRatio_0 = 0 \tag{3.2}$$

[12] Der Begriff „Größenverhältnis-Wert“ wird zum Teil auch synonym mit „Größenverhältnis“ benutzt. Zwar kann *ein* Wert allein kein Verhältnis aufstellen, jedoch wird davon ausgegangen, dass ein solcher Wert *immer* im Zusammenhang mit den anderen Größenverhältnis-Werten steht.

Auswertung von Sichtbarkeit und Zoom: Der Zoom verschiebt den Index von $BeltRes_i$ additiv um den in $ZoomIndex$ angegebenen Wert, d.h. der Ring i wird nicht in seiner eigenen Auflösung ($BeltRes_i$) dargestellt, sondern in der Auflösung des $i + ZoomIndex$.ten Ringes ($BeltRes_{i+ZoomIndex}$).[13] Im Zusammenhang mit (3.1) ergibt sich die Maximalvergrößerung bei $ZoomIndex = 0$ und die Maximalverkleinerung bei $ZoomIndex = NumOfBelts - 1$. Alle Ringe j mit $j < ZoomIndex$ können *nicht* dargestellt werden. Da für die eigentliche Berechnung des FISHEYE–VIEWS nur die Parameter der wirklich sichtbaren Ringe von Bedeutung sind, werden nun Größenverhältnisse und Auflösungen für die sichtbaren Ringe zusammengefasst.

$$NumOfVisBelts = \sum_{i=0}^{\substack{NumOfBelts\\ -ZoomIndex}} BeltVis_i \tag{3.3}$$

$\Rightarrow$ Anzahl der sichtbaren Ringe

$$VisBeltRes_{k=j-\sum_{i=0}^{j} \neg BeltVis_i} = BeltRes_{j+ZoomIndex} \tag{3.4}$$

wobei gilt $0 \leq j < NumOfBelts - ZoomIndex$

$\Rightarrow$ Auflösung des sichtbaren Ringes k

$$VisBeltRatio_{k=j-\sum_{i=0}^{j} \neg BeltVis_i} = BeltRatio_j \tag{3.5}$$

wobei gilt $0 \leq j < NumOfBelts - ZoomIndex$

$\Rightarrow$ Größenverhältnis-Wert des sichtbaren Ringes k

Es ist leicht zu sehen, dass für den Index von $VisBeltRatio_k$ gilt:

$$0 \leq k < NumOfVisBelts$$

Nach der Auswertung des Zooms, unterliegen die in der Fokusregion dargestellten Bilddaten einer neuen Auflösung. Man setze in (3.4) $j = 0$ (also der nullte Ring $\Rightarrow$ Fokusregion) und erhält $k = 0$, so dass

$$VisBeltRes_0 = BeltRes_{ZoomIndex}$$

gilt. Die Ratio der Fokusregion ist somit nach (3.5) und (3.2) weiterhin 0, denn[14]

$$VisBeltRatio_0 \overset{(3.5)}{=} BeltRatio_0 \overset{(3.2)}{=} 0$$

Vorbereitung zur Einteilung der Größenverhältnisse: Es sei

$$SumOfRatio = \sum_{k=1}^{NumOfVisBelts-1} VisBeltRatio_k \tag{3.6}$$

[13] Man beachte, dass die Indizes von $BeltVis_i$ und $BeltRatio_i$ nicht verschoben werden. Dies sichert visuelle Ähnlichkeit für die FISHEYE–VIEWS vor und nach einer Zoom Operation.

[14] Es sei angemerkt, dass die Ratio der Fokusregion keinen Einfluß hat. Dies sieht man auch in den Formeln von (3.6) und (3.8), die erst bei Index 1 beginnen und somit die Fokusregion außer acht lassen.

die Summe über die Größenverhältnis-Werte aller sichtbaren Ringe (außer der Fokusregion). Der Wert $VisBeltRatio_k$ hat die Bedeutung, dass der Ring k auf dem Bildschirm in seiner Breite $\frac{VisBeltRatio_k}{SumOfRatio}$ des vorhandenen Platzes füllt.[15] Diese auf die Bildschirmdarstellung bezogenen Größenverhältnisse müssen umgerechnet werden auf die Verhältnisse der Ringe im Ausgangsbild. Hierbei spielt die Auflösung eines Ringes die entscheidende Rolle. Die Breite eines Ringes im FishEye–View ist im Ausgangsbild um den Faktor der Auflösungzahl des Ringes größer.[16] Also werden die Größenverhältnisse in der Form

$$VisBeltRatioRes_k = VisBeltRatio_k \cdot VisBeltRes_k \tag{3.7}$$

umgerechnet. Demnach ergibt sich auch eine Summe über die Größenverhältnisse im Ausgangsbild:

$$SumOfRatioRes = \sum_{k=1}^{NumOfVisBelts-1} (VisBeltRatio_k \cdot VisBeltRes_k) \tag{3.8}$$

Vorbereitung der Berechnung: Um die geometrische Struktur eines *bereits berechneten* FishEye–Views mathematisch kurz und prägnant darzustellen, bedarf es keiner komplizierten Parameter. Abbildung 3.3 verdeutlicht, dass einzig die umrandenden Rechteckkoordinaten der Ringe benötigt werden, um den FishEye–View zu zeichnen. Die Berechnung des FishEye–Views soll also folgende Koordinaten bestimmen:

$$\begin{array}{lll} vBeltCoord_{r,k} & \Rightarrow & \text{Koordinate des Ringes } k \text{ in Richtung r} \\ & & \text{im Ausgangsbild} \\ & & \text{wobei gilt } r \in \{left, top, right, bottom\} \\ sBeltCoord_{r,k} & \Rightarrow & \text{Koordinate des Ringes } k \text{ in Richtung r} \\ & & \text{auf dem Bildschirm} \\ & & \text{wobei gilt } r \in \{left, top, right, bottom\} \end{array}$$

Startpunkt und Endpunkt der Berechnung: Abbildung 3.3 zeigt, dass die Ringe im FishEye–Viewvon innen nach außen gezählt werden. Die Fokusregion ist also der nullte Ring($k = 0$). Der letzte Ring hat den Index $k =$

[15]**Beispiel:** Gegeben sei ein FishEye–View mit einem 2er, 4er und 8er Ring, deren Ratios sich 1:1:1 zueinander verhalten. Dann sind alle 3 Ringe auf dem Bildschirm gleich breit. Verhalten sich die Ratios jedoch 1:2:1 zueinander, so nimmt der 4er Ring die Hälfte des zur Verfügung stehenden Platzes in alle Richtungen ein, während die anderen beiden Ringe nur jeweils ein Viertel belegen.

[16]**Beispiel:** Gegeben sei ein FishEye–View mit einem 2er, 4er und 8er Ring, deren Ratios sich 1:1:1 zueinander verhalten. Die Breite jedes Ringes auf dem Bildschirm in eine Richtung (also z.B. links vom Fokus) betrage 100 Pixel. Damit nimmt der 2er Ring im Ausgangsbild eine Breite von 200 Pixeln, der 4er Ring von 400 Pixeln und der 8er Ring von 800 Pixeln ein. Die Ratios 1:1:1 auf dem Bildschirm bedeuten also im Ausgangsbild 2:4:8.

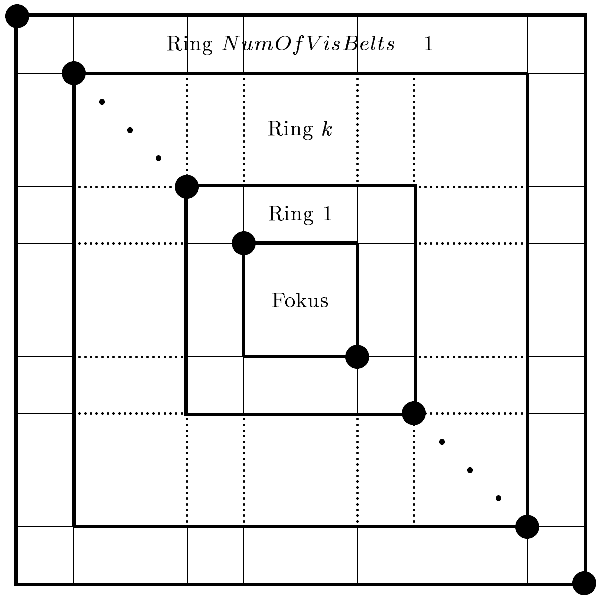


Abbildung 3.3: Kurzbeschreibung der FishEye-Geometrie. Die mit einem großen Punkt gekennzeichneten Eckpunkte der Kontextringe genügen zur Beschreibung eines bereits berechneten FishEye–Views. Dies gilt sowohl für das Ausgangsbild als auch für das Bildschirmbild.

$NumOfVisBelts - 1$. Die Koordinaten für beide Ringe sind bereits gegeben:[17]

$$\begin{aligned}
vBeltCoord_{left,0} &= Left1 \\
vBeltCoord_{top,0} &= Top1 \\
vBeltCoord_{right,0} &= Right1 \\
vBeltCoord_{bottom,0} &= Bottom1 \\
vBeltCoord_{left,NumOfVisBelts-1} &= 0 \\
vBeltCoord_{top,NumOfVisBelts-1} &= 0 \\
vBeltCoord_{right,NumOfVisBelts-1} &= vPicWidth - 1 \\
vBeltCoord_{bottom,NumOfVisBelts-1} &= vPicHeight - 1
\end{aligned}$$

Die Berechnung des FishEye–Views: Es bleiben also noch die Eckkoordinaten der Ringe $k = 1, \ldots, NumOfVisBelts - 2$ zu berechnen. Dazu muss der Platz zwischen dem Fokus und dem Rand des Ausgangsbildes auf die Ringe $k = 1, \ldots, NumOfVisBelts - 1$ nach den durch $VisBeltRatio_k$ vorgegebenen Größenverhältnissen aufgeteilt werden. Die Breite eines Ringes in eine bestimmte Richtung berechnet sich nach (3.7) und (3.8) wie folgt:

$$\begin{aligned}
vBeltWidth_{r,k} &= |vBeltCoord_{r,NumOfVisBelts-1} - vBeltCoord_{r,0}| \\
&\quad \cdot \frac{VisBeltRatioRes_k}{SumOfRatioRes}
\end{aligned} \tag{3.9}$$

[17] Es sei darauf hingewiesen, dass damit nicht die Größe des äußeren Ringes gegeben ist, sondern nur seine nach außen begrenzenden Koordinaten. Die Größe wird erst durch die Koordinaten des nächstinneren Ringes bestimmt. Dies ist bei allen Ringen außer der Fokusregion so.

Die Breite des Ringes k nutzt man nun zur Berechnung seiner Koordinaten auf Basis der (bereits berechneten) Koordinaten des nächstinneren Ringes $k-1$ wie folgt:

$$vBeltCoord_{r,k} = \begin{cases} vBeltCoord_{r,k-1} - vBeltWidth_{r,k} & r \in \{left, top\} \\ vBeltCoord_{r,k-1} + vBeltWidth_{r,k} & r \in \{right, bottom\} \end{cases} \tag{3.10}$$

Nachbereitung der Berechnung: Hat man alle Eckpunktkoordinaten der Ringe des FishEye–View im Ausgangsbild, so ist es leicht, diese unter Beachtung der Auflösungen der Ringe auf Bildschirmkoordinaten umzurechnen:

$$sBeltCoord_{left,NumOfVisBelts-1} = 0 \tag{3.11a}$$

$$sBeltCoord_{top,NumOfVisBelts-1} = 0 \tag{3.11b}$$

$$\begin{aligned} sBeltCoord_{r,k} &= sBeltCoord_{r,k+1} + \frac{vBeltWidth_{r,k+1}}{VisBeltRes_{k+1}} \\ &\quad \text{für } r \in \{left, top\} \end{aligned} \tag{3.12}$$

$$\begin{aligned} sBeltCoord_{right,0} &= sBeltCoord_{left,0} - 1 \\ &\quad + \frac{vBeltCoord_{right,0} - vBeltCoord_{left,0} + 1}{VisBeltRes_0} \end{aligned} \tag{3.13a}$$

$$\begin{aligned} sBeltCoord_{bottom,0} &= sBeltCoord_{top,0} - 1 \\ &\quad + \frac{vBeltCoord_{bottom,0} - vBeltCoord_{top,0} + 1}{VisBeltRes_0} \end{aligned} \tag{3.13b}$$

$$\begin{aligned} sBeltCoord_{r,k} &= sBeltCoord_{r,k-1} + \frac{vBeltWidth_{r,k}}{VisBeltRes_k} \\ &\quad \text{für } r \in \{right, bottom\} \end{aligned} \tag{3.14}$$

Die Darstellung des FishEye–Views auf dem Bildschirm geschieht demnach so, dass basierend auf den in (3.10) berechneten Koordinaten die Bilddaten aus dem Ausgangsbild extrahiert und in korrekter Auflösung auf den korrespondierenden Bildschirmkoordinaten dargestellt werden. Auf diese Art wird jedes durch den FishEye–View erzeugte Rechteck mit einheitlicher Auflösung einzeln behandelt.

Nebenbedingung während der Berechnung: Bereits bekannt ist, dass die Basis dieser Darstellungstechnik ein Ausgangsbild in voller Auflösung (o.B.d.A. 1:1) ist. Ein Teil davon wird gewöhnlich (außer bei Zoom[18]) in der Fokusregion

[18] Bei Zoom wird in der Fokusregion eine höherzahlige Auflösung dargestellt. Die folgenden Ausführungen sind zum größten Teil auch auf diesen Fall anwendbar. Es gibt jedoch einen Fall, in dem die höherzahlige Auflösung im Fokus eine Lösung des Teilerproblems verhindert. Dieser Fall wird gesondert behandelt.

dargestellt. Alle anderen Rechtecke im FISHEYE–VIEW sind durch verschiedene Auflösungskombinationen gleichmäßig oder ungleichmäßig skalierte Regionen des Ausgangsbildes. Nun sind diese Regionen in ihrer Höhe bzw. Breite nicht immer ganzzahlig durch die jeweilige Auflösung teilbar[19](folgt man zumindest der Berechnungsvorschrift in (3.9)). Unter der Verwendung verschiedener Skalierungsalgorithmen ergibt dies verschiedene Probleme:

1. Man skaliert unter exakter Einhaltung der Auflösung und der Zielpixelzahl. Dies hat zur Folge, dass die Bildinformation ganzer Pixelzeilen bzw. -spalten des Ausgangsbildes im Zielbild nicht vorhanden ist. Das Maximum verlorener Zeilen bzw. Spalten beträgt $VisBeltRes_k - 1$. Bei hoher Auflösungszahl kann es somit zu starken Sprüngen an den Rechteckgrenzen kommen. Vorteil dieser Herangehensweise sind jedoch die schnellen Algorithmen.

2. Man skaliert unter exakter Einhaltung von Ausgangs- und Zielpixelzahl. Dies führt zu einer Verfälschung der Auflösung, die damit innerhalb eines Ringes i.a. *nicht* mehr homogen ist, da die Breite eines Ringes in alle vier Richtungen unterschiedlich ist und somit auch die Verfälschung. Die Abweichung von der eigentlichen Auflösung beträgt:

$$\Delta_{absolut} = \frac{APZ}{ZPZ} - Res \tag{3.15a}$$

$$\Delta_{relativ} = \frac{APZ}{ZPZ \cdot Res} \tag{3.15b}$$

$$\begin{aligned} \text{mit} \quad & APZ = vBeltWidth_{r,k} \\ \Rightarrow \quad & \text{Ausgangspixelzahl} \\ & Res = VisBeltRes_k \\ \Rightarrow \quad & \text{gewünschte Auflösung} \\ & ZPZ = \left\lfloor \frac{APZ}{Res} \right\rfloor \\ \Rightarrow \quad & \text{Zielpixelzahl}^{20} \end{aligned}$$

Die maximale Abweichung erhält man, wenn APZ bei der Division den größten Rest läßt (also APZ ist Element der Teilerklasse $[Res - 1]$ von Res):

$$APZ = (ZPZ + 1)\, Res - 1$$

Es folgt damit die maximale Abweichung nach (3.15):

$$\hat{\Delta}_{absolut} = \frac{APZ}{\lfloor \frac{APZ}{Res} \rfloor} - \frac{APZ + 1}{\lfloor \frac{APZ}{Res} \rfloor + 1} \tag{3.16a}$$

$$\hat{\Delta}_{relativ} = \frac{APZ \left(\lfloor \frac{APZ}{Res} \rfloor + 1 \right)}{\lfloor \frac{APZ}{Res} \rfloor (APZ + 1)} \tag{3.16b}$$

[19]**Beispiel:** Gegeben ist ein Rechteck mit einer Breite von 14 Pixeln und dieses soll mit einer horizontalen Auflösung von 1:4 dargestellt werden. 14 ist nicht ganzzahlig durch 4 teilbar. 3,5 Pixel müssten im Zielbild dargestellt werden.

[20]$\lfloor a \rfloor$ liefert für $a > 0$ den ganzzahligen Teil von a.

> Der Vorteil dieses Ansatzes besteht darin, dass alle Pixel des Ausgangsbildes zur Berechnung des FISHEYE–VIEWS herangezogen werden. Nachteile sind die Auflösungsverfälschung und die eher langsamen Algorithmen, die diese Art der Skalierung umsetzen.[21]

Beide Ansätze bieten erstrebsame Vorteile und nicht tolerierbare Nachteile. Das Problem der ganzzahligen Teilbarkeit läßt sich durch diese Ansätze jedenfalls nicht befriedigend lösen. Es ist jedoch möglich, für die Berechnung der Breite eines Ringes in (3.9) eine Nebenbedingung zu finden, die $vBeltWidth_{r,k}$ so variiert, dass eine restlose Teilung durch $VisBeltRes_k$ möglich ist. Zwar hat eine solche Bedingung den Nachteil, dass die in $VisBeltRatio_k$ spezifizierten Größenverhältnisse nicht genau eingehalten werden, aber dafür kann man nun unabhängig vom verwendeten Skalierungsalgorithmus eine Auflösungsverzerrung oder verlorene Pixelzeilen bzw- spalten ausschließen.

Die Idee beruht darauf, dass die Berechnung des FISHEYE–VIEWS von innen nach außen über die einzelnen Ringe iteriert. Dabei ist es die Aufgabe, den noch nach außen hin vorhandenen Platz so einzuteilen, dass die Breite des aktuell zu berechnenden Ringes k durch dessen Auflösung und der übrigbleibende Platz durch die Auflösung des nächsten Ringes teilbar ist. Daraus ergibt sich folgende Nebenbedingung zu (3.9):

$$vBeltWidth_{r,k} \equiv 0 \pmod{VisBeltRes_k} \tag{3.17a}$$

$$(vBeltDist_{r,k} - vBeltWidth_{r,k}) \equiv 0 \pmod{VisBeltRes_{k+1}} \tag{3.17b}$$

$$vBeltDist_{r,k} \geq vBeltWidth_{r,k} \geq 0 \tag{3.17c}$$

Hierbei sei

$$k = 1, \ldots, NumOfVisBelts - 1$$

und

$$vBeltDist_{r,k} = |vBeltCoord_{r,NumOfVisBelts-1} - vBeltCoord_{r,k-1}|$$

Die Bedingung (3.17a) stellt die Forderung auf, dass die Breite eines Ringes durch seine Auflösung teilbar ist. (3.17c) sichert ab, dass Ring k nicht mehr Platz einnimmt, als überhaupt noch zum Rand hin vorhanden ist (Abbildung 3.4).

Die Forderung (3.17b) soll im folgenden näher betrachtet werden. Sie ermöglicht, dass der Platz, der *nach* der Berechnung von Ring k noch übrigbleibt, durch die Auflösung des nächsten Ringes teilbar ist. Dies garantiert für die nächste Iteration, dass wieder eine entsprechende Einteilung gefunden werden kann.

Satz 1. *Sei $vBeltDist_{r,k}$ durch $VisBeltRes_k$ ganzzahlig teilbar.*
Dann finden sich ein a und ein b mit $a + b = vBeltDist_{r,k}$, so dass gilt:

$$VisBeltRes_k | a \qquad und \qquad VisBeltRes_{k+1} | b$$

[21] Es wird von Algorithmen ausgegangen, die die Skalierung nicht über das Weglassen von Zeilen bzw. Spalten erreichen, sondern korrekt mittels Filtertechnik und Floating-Point-Arithmetik skalieren.

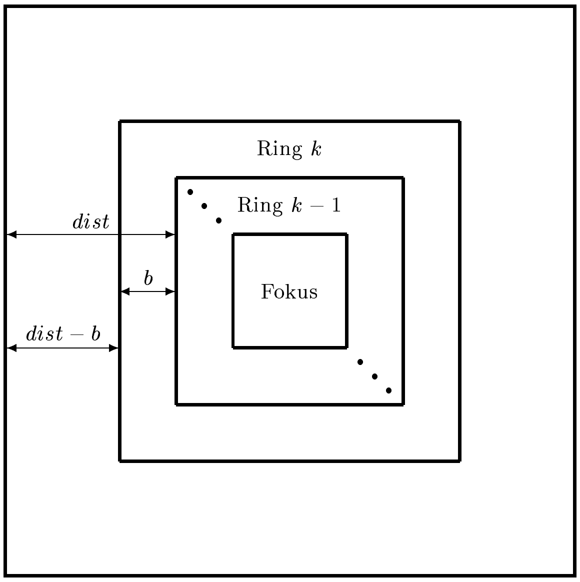


Abbildung 3.4: Aufteilung des FISHEYE–VIEWS während der k-ten Iteration. Die Bedingung (3.17) besagt, dass b durch die Auflösung von Ring k und $dist-b$ durch die Auflösung von Ring $k-1$ teilbar sein müssen. Weiterhin darf b nicht größer als $dist$ werden und beide nicht kleiner als 0. Hierbei seien $dist = vBeltDist_{r,k}$ und $b = vBeltWidth_{r,k}$.

Beweis.

$$\begin{aligned}
vBeltDist_{r,k} &= n \cdot VisBeltRes_k \\
&= m_1 \cdot VisBeltRes_k + m_2 \cdot VisBeltRes_k \\
&\quad \text{mit } 1 \leq m_1 \leq n \quad \text{und} \quad 0 \leq m_2 \leq n-1 \\
&= m_1' \cdot VisBeltRes_k + m_2' \cdot \mathbf{kgV}(VisBeltRes_k, VisBeltRes_{k+1}) \\
&= m_1' \cdot VisBeltRes_k + m_2'' \cdot VisBeltRes_{k+1}
\end{aligned}$$

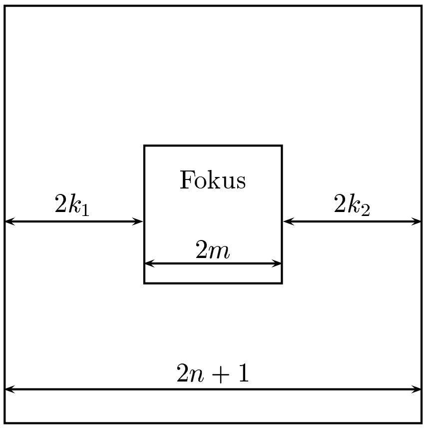
□

Das Ergebnis $vBeltWidth_{r,k}$ aus (3.9) wird solange schrittweise um 1 erhöht, bis es der Nebenbedingung (3.17) genügt.

Für den Fokus muss (3.17) leicht abgewandelt werden, da weder $vBeltWidth_{r,k}$ noch $vBeltDist_{r,k}$ für $k = 0$ definiert sind. Zudem werden die Koordinaten des Fokus nicht berechnet, sondern sie sind gegeben. Dennoch müssen die Größe und Position des Fokus so angepasst werden, dass Breite und Höhe problemlos durch die Auflösung des Fokus und der nach außen hin verbleibende Platz durch die Auflösung des ersten Ringes teilbar sind. Nur dann kann die Voraussetzung von Satz 1 und damit die Nebenbedingung für alle Ringe erfüllt werden. Die Nebenbedingung für den Fokus wird also wie folgt definiert:

$$(vBeltCoord_{right,0} - vBeltCoord_{left,0} + 1) \equiv 0 \pmod{VisBeltRes_0} \quad (3.18a)$$

$$(vBeltCoord_{bottom,0} - vBeltCoord_{top,0} + 1) \equiv 0 \pmod{VisBeltRes_0} \quad (3.18b)$$



(a) $(2n+1) \neq 2k_1 + 2m + 2k_2$

Abbildung 3.5: Fall der Nichterfüllbarkeit der Nebenbedingung für den Fokus.

$$vBeltDist_{r,1} \equiv 0 \pmod{VisBeltRes_1} \tag{3.18c}$$

$$vBeltDist_{r,1} \geq 0 \tag{3.18d}$$

Die Bedingungen (3.18a) und (3.18b) stellen lediglich die Forderung auf, dass Breite und Höhe des Fokus durch seine Auflösung teilbar sein sollen, während (3.18d) garantiert, dass der Fokus nicht größer als das Ausgangsbild selbst werden kann. Forderung (3.18c) ermöglicht (analog zu (3.17b)), dass der Platz außerhalb des Fokus in alle Richtungen durch die Auflösung des ersten Ringes teilbar ist und garantiert damit die Anwendbarkeit der Nebenbedingung auf die Ringe $k = 1, \ldots, NumOfVisBelts - 1$.

Um die Nebenbedingung für den Fokus zu erfüllen, müssen die eigentlich vorgegebenen Fokuskoordinaten mit möglichst geringer Veränderung angepasst werden. Dazu genügt es, den Fokus in eine Richtung (also z.B. nach links bzw. nach oben) schrittweise um einen Pixel zu verschieben und anschließend in die jeweils entgegengesetzte Richtung (also z.B. nach rechts bzw. nach unten) zu erweitern. Es ist leicht zu sehen, dass dieses Verfahren immer eine Lösung bietet, wenn $ZoomIndex = 0$ gilt, also die Auflösung des Fokus 1:1 beträgt.

Für den Fall $ZoomIndex > 0$ treten jedoch Komplikationen auf. Es kann dann vorkommen, dass es keine nach (3.18) gültige Größe und Position des Fokus gibt. Dies ist dann der Fall, wenn die Weite bzw. Höhe des Ausgangsbildes ungerade und die Auflösungen des Fokus sowie des ersten Ringes gerade sind. Somit müssten auch die Ausmasse des Fokus und des ersten Ringes gerade sein. Die Summe von geraden Zahlen kann jedoch niemals eine ungerade Zahl erzeugen, was nötig wäre, um das Ausgangsbild in Weite bzw. Höhe vollständig aufzuteilen. Die Abbildung 3.5 verdeutlicht den Zusammenhang. Wenn es in der Praxis zu diesem Problem kommt, dann wird (3.18c) höher priorisiert, um die Voraussetzungen für die Nebenbedingung der anderen Ringe zu gewährleisten.

Die Auswirkungen der Nebenbedingung für den Fokus sind in den Abbildungen 3.6a bis 3.6c zu sehen. Man erkennt darin den Zusammenhang zwischen der Position des Fokus und der Auflösung des ersten Ringes. Dies ist die direkte Auswirkung der Bedingung (3.18c).

Funktionsdarstellungen der Berechnungsvorschrift: Die gesamte Berechnung des FishEye–Views läuft darauf hinaus, ein Ausgangsbild unter Beachtung von Randparametern in einer bestimmten Weise einzuteilen und diese Einteilung auf Bildschirmkoordinaten umzurechnen. Zum Verständnis der Berechnungsvorschrift ist es sicherlich sinnvoll, einige Parameter und berechnete Werte zueinander in Verbindung zu setzen:

1. **Position des Fokus im Ausgangsbild und auf dem Bildschirm.** Hier wird die direkte Funktion der Berechnung gezeigt : Welche Position des Fokus im Ausgangsbild erzeugt welche Bildschirmposition. Dies ist natürlich abhängig von den verwendeten Kontextauflösungen, Zoomstufen und Größenverhältnissen. Abbildung 3.7 verdeutlicht die Zusammenhänge unter Verwendung verschiedener Randparameter.

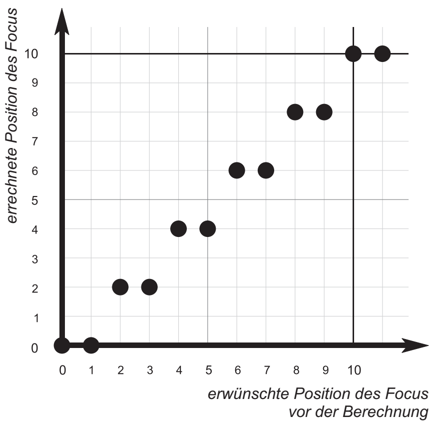
2. **Platzersparnis durch den FishEye–View.** Ein wesentlicher Vorteil des FishEye–Views ist seine platzsparende Darstellung im Vergleich zum Ausgangsbild. Da es sich hierbei um eine besonders wichtige Eigenschaft handelt, sollen die Abhängigkeiten einzeln betrachtet werden.

   (a) *Platzersparnis mit verschiedenen Sichtbarkeitskonfigurationen.* Abbildung 3.8a gibt einen Eindruck von der Platzersparnis unter Verwendung verschiedener Kontextringe. Der FishEye–Viewbenötigt dann weniger Platz, wenn mehr Kontextringe mit höherzahligen und weniger Ringe mit niederzahligen Grundauflösungen dargestellt werden. Die Ringe können durch ihre Sichtbarkeit ein- bzw. ausgeschaltet werden.

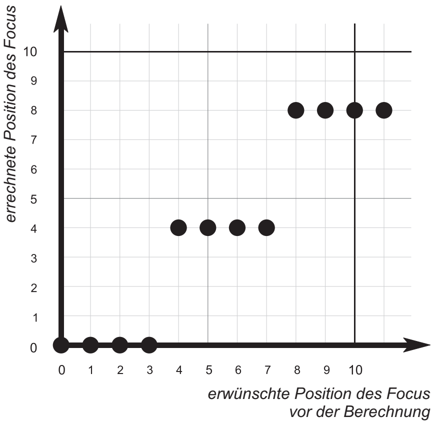
   (b) *Platzersparnis mit verschiedenen Kontextauflösungen.* Abbildung 3.8b gibt einen Eindruck von der Platzersparnis unter Verwendung verschiedener Kontextauflösungen. Es zeigt sich, dass FishEye–Views mit höherzahligen Kontextauflösungen weniger Platz beanspruchen.

   (c) *Platzersparnis mit verschiedenen Größenverhältnissen.* Abbildung 3.8c gibt einen Eindruck von der Platzersparnis unter Verwendung verschiedener Größenverhältnisse. FishEye–Views, die höherzahlige Kontextauflösungen mittels erhöhter Ratiowerte priorisieren, beanspruchen weniger Platz.

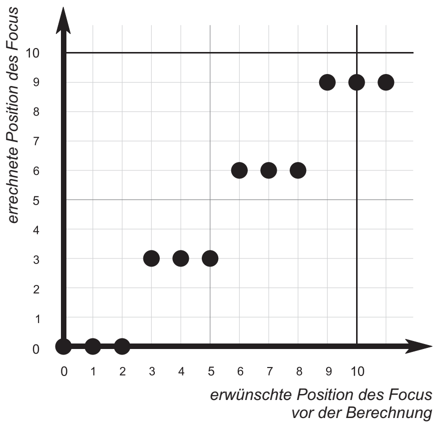
   (d) *Platzersparnis mit verschiedenen Zoomstufen.* Abbildung 3.8d gibt einen Eindruck von der Platzersparnis unter Verwendung verschiedener Zoomstufen. Wie nicht anders zu erwarten war, beanspruchen FishEye–Views mit höheren Zoomstufen weniger Platz.



(a) Grundauflösungen 1, 2, 4 und 8.
Kein Zoom.



(b) Grundauflösungen 1, 2, 4 und 8.
Zoom von einer Stufe.



(c) Grundauflösungen 1, 3 und 6.
Kein Zoom.

Abbildung 3.6: Auswirkung der Nebenbedingung auf die Position des Fokus bei FishEye–Views mit verschiedenen Grundauflösungen und Zoomstufen. Die Werte beziehen sich auf die linke Koordinate des Fokus im Ausgangsbild.

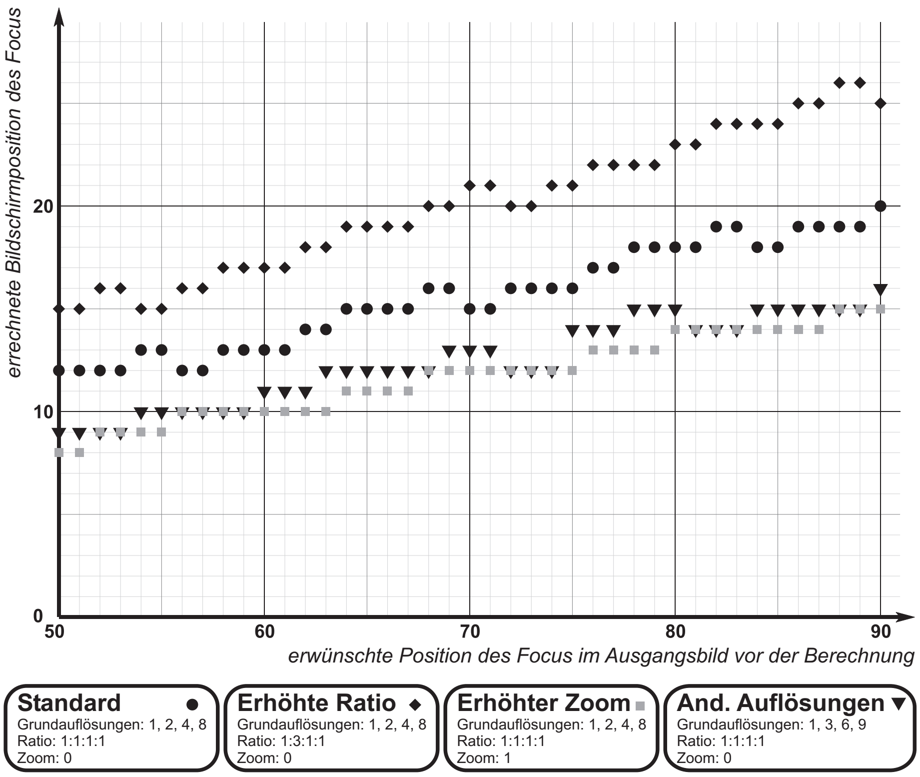
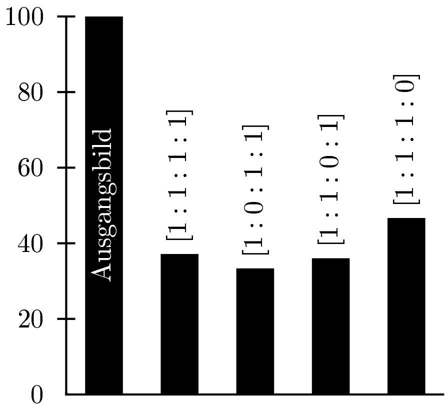
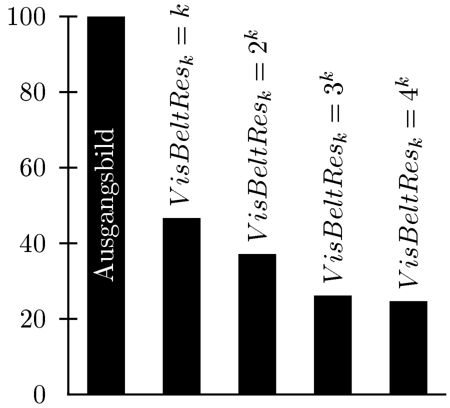


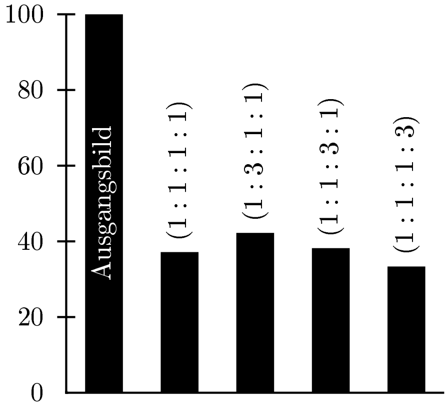
Abbildung 3.7: Position des Fokus im Ausgangsbild und auf dem Bildschirm. Die Werte beziehen sich auf die linke Koordinate des Fokus im jeweiligen System.



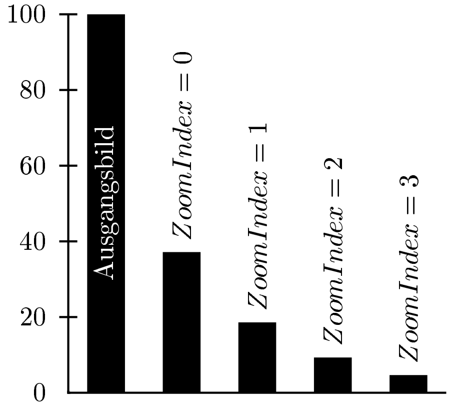
(a) Verschiedene Sichtbarkeitskonfigurationen bei einem FishEye–View mit 2er, 4er und 8er Ring. Die Sichtbarkeit ist in eckigen Klammern angegeben, wobei die erste Zahl für den Fokus und die letzte für den 8er Ring steht.



(b) Verschiedene Kontextauflösungen bei FishEye–Views mit jeweils 3 sichtbaren Ringen.



(c) Verschiedene Größenverhältnisse bei einem FishEye–View mit 2er, 4er und 8er Ring. Die Verhältnisse sind in runden Klammern angegeben, wobei die erste Zahl für den Fokus und die letzte für den 8er Ring steht.



(d) Verschiedene Zoomstufen bei einem FishEye–View mit 2er, 4er, 8er, 16er, 32er und 64er Ring. Es sind jedoch nur die ersten 3 Ringe sichtbar. In allen Zoomstufen werden also 3 Kontextringe angezeigt.

Abbildung 3.8: Platzersparnis durch den FishEye–View. Angaben in Prozent. Bei allen Berechnungen lag ein Ausgangsbild in der Größe 10000 × 10000 Pixel mit einem Fokus von 2002 × 2002 Pixeln zugrunde. Die Platzersparnis bezieht sich somit auf die Höhe bzw. Breite. Diese Aussagen gelten für alle FishEye–Views mit gleichem Verhältnis zwischen den Größen des Fokus zu denen des Ausgangsbildes.

3.5 Die Interaktion mit dem rechteckigen Fish-Eye–View

In diesem Abschnitt sollen die Interaktionsmöglichkeiten mit dem monofokalen FishEye–View vorgestellt werden. Interaktion soll den Nutzer in die Lage versetzen, mit Hilfe der Darstellungstechnik das vorhandene Bildmaterial zu begreifen und zu erkunden. Es ist von enormer Wichtigkeit, dass auf jede Aktion des Nutzers schnellstmöglich eine Reaktion des Systems folgt. Nur so ist es möglich, ein *Gefühl* für die verzerrte Darstellungsweise des FishEye–Views zu erhalten.

Zunächst sollen die Techniken vorgestellt werden, mit deren Hilfe die Parameter der Darstellung angepasst werden können. Im weiteren müssen einige mathematische Probleme beleuchtet werden, die durch die besondere Situation während der Interaktion entstehen. Am Ende dieses Abschnitts wird das Interaktivitätsmodell des FishEye–Views vorgestellt, das die verschiedenen Techniken unter der Maßgabe kurzer Antwortzeiten miteinander verknüpft.

3.5.1 Techniken zur Parametermanipulation

Folgende Interaktionstechniken sind allgemein üblich und wurden umgesetzt:

- Panning
- Resize
- Jump
- View Panning

Im folgenden soll auf jede dieser Techniken näher eingegangen werden.

Panning: Hierbei wird der Fokus schrittweise auf der Darstellungsfläche verschoben. Der Nutzer kann also seinen speziellen Interessen folgend verschiedene Abschnitte im Detail erkunden.

Die Nebenbedingung (3.18) für den Fokus (vorgestellt in Abschnitt 3.4) erlaubt jedoch nicht, dass der Fokus sich an allen Stellen im Ausgangsbild befinden kann (siehe auch Abbildung 3.4 auf Seite 43). Dennoch kann sich der Fokus an allen möglichen Positionen innerhalb der Darstellungsfläche des FishEye–Viewsbefinden. Da diese Fläche kleiner als die des Ausgangsbildes ist, kann man schon rein mathematisch keine Funktion erwarten, die jede Position im Ausgangsbild auf eine andere Position innerhalb der Darstellungsfläche abbildet. Die Menge der möglichen Positionen für den Fokus ist im FishEye–View also wesentlich kleiner als im Ausgangsbild. Die sich hieraus ergebenden mathematischen Probleme werden in 3.5.2 näher beleuchtet. Es genügt jedoch an dieser Stelle zu sagen, dass das visuelle Empfinden dadurch nicht gestört wird, da der Fokus jede Position auf der Darstellungsfläche einnehmen kann.

Durch das Verschieben des Fokus ändert sich die Größe der Gesamtdarstellung prinzipiell nicht, da Größe des Fokus und Kontextierungsparameter konstant bleiben.[22] Diese Interaktionstechnik erfordert *während* der Ausführung ein ständiges Neuzeichnen des FishEye–Views.

Resize: Diese Interaktionstechnik ermöglicht es dem Benutzer, die Größe des FishEye–Views einzustellen. Im Zusammenhang mit dem Panning wird somit die Forderung nach exakter Definitionsmöglichkeit der Fokusregion erfüllt.

Auch beim Resize spielt die Nebenbedingung für den Fokus eine entscheidende Rolle. Es gelten obige Ausführungen.

Durch die Änderung der Größe des Fokus wird natürlich auch der gesamte Platzbedarf des FishEye–Views größer bzw. kleiner. Diese Interaktionstechnik erfordert *während* der Ausführung ein ständiges Neuzeichnen des FishEye–Views.

Jump: Mit Hilfe dieser Technik kann der Fokus augenblicklich an eine neue Stelle im Ausgangsbild gesetzt werden, ohne dabei verschoben werden zu müssen. Die Koordinaten im Ausgangsbild, an die der Fokus gesetzt wird, entsprechen visuell den zugehörigen Koordinaten auf der Darstellungsfläche, die der Nutzer mit der Maus spezifiziert. Im Gegensatz zum Panning wird der Fokus also nicht an eine neue Stelle im FishEye–View gesetzt, von deren Bildinformation der Nutzer noch keine Vorstellung hat. Die Technik des Jump ist eine elegante Art, die den Nutzer interessierenden Regionen des Bildes direkt anzuspringen.

Es gibt zwei Möglichkeiten für diese Interaktion:

1. **Versetzen ohne Größenänderung.** Hierbei wird der Fokus in seiner bisherigen Größe an eine neue Stelle im Ausgangsbild gesetzt. Referenzpunkt ist die Mitte des Fokus.

2. **Versetzen mit Größenänderung.** Der Nutzer spezifiziert zwei neue Punkte, die den Fokus aufspannen. Dies ermöglicht, Position und Größe des Fokus gleichzeitig zu definieren.

Natürlich gilt auch bei dieser Technik die Nebenbedingung für den Fokus. Es gelten die Aussagen zum Panning.

Die Interaktionstechnik des Jump benötigt im Gegensatz zum Panning und Resize *während* der Ausführung der Aktion keine Neuzeichnung des FishEye–Views. Erst nachdem der neue Fokus spezifiziert ist, wird der FishEye–View neu berechnet und dargestellt. Falls neben der Position auch die Größe des Fokus verändert wird, kommt es zwangsläufig auch zu einer Änderung des Platzbedarfs der Darstellung.

View Panning: Es wird ermöglicht, die Gesamtdarstellung des FishEye–Views zu verschieben. Im Zusammenhang mit der Möglichkeit des Zoom integriert diese Technik einen Zooming-und-Panning-Ansatz, wie er aus vielen Anwendungen bekannt ist. Falls sogar alle Kontextringe im FishEye–View ausgeschaltet (nicht sichtbar) sind, verhält sich das Interaktionssystem vollständig wie beim Zooming-und-Panning.

[22] Eine leichte Veränderung der Größe geschieht höchstens dadurch, dass die Breiten der Ringe durch den Einfluß von Integerarithmetik nicht genau den zugrundeliegenden Parameterwerten entsprechen. Wird der Fokus verschoben, so kommt es zu einer neuen Aufteilung der Ringe und somit zu anderen Rundungsfehlern.

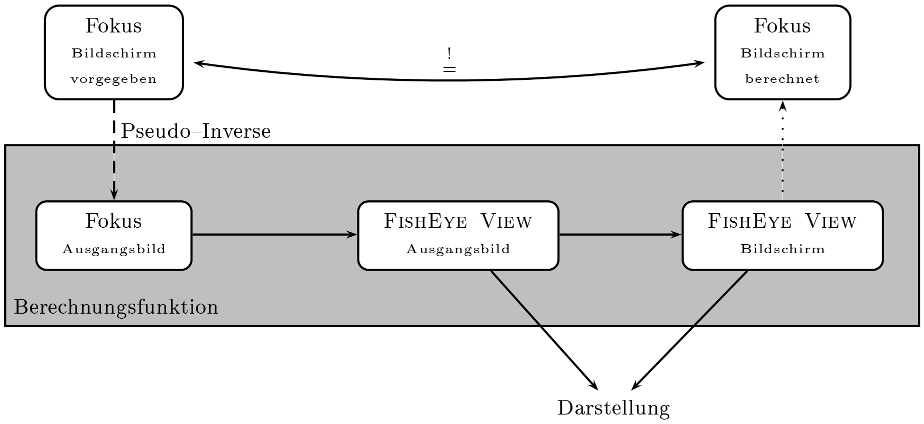


Abbildung 3.9: Berechnungsweg während der Interaktion. Die grau unterlegte Sektion beschreibt die normale Berechnungsfunktion des FISHEYE–VIEWS. Während der Interaktion werden jedoch vorher mittels einer Pseudo–Inversen die Bildschirmkoordinaten des Fokus in Koordinaten im Ausgangsbild umgerechnet. Nach der Berechnung kann man die *neu berechneten* Bildschirmkoordinaten des Fokus mit den vorgegebenen vergleichen. Ziel ist es, die Pseudo–Inverse so zu definieren, dass beide gleich sind.

Die Ausführung des View Panning erfordert weder während noch nach der Interaktion eine Neuberechnung des FISHEYE–VIEWS.

3.5.2 Mathematische Probleme

Es ist bekannt, dass i.a. die Größe des FISHEYE–VIEWS wesentlich geringer ist als die des Ausgangsbildes. Gerade das ist einer der wesentlichen Vorteile dieser Darstellungstechnik. Jedoch ergibt sich daraus das Problem, dass es zu der Position des Fokus auf dem Bildschirm *mehrere* mögliche Positionen im Ausgangsbild gibt. Die Funktion zur Berechnung des FISHEYE–VIEWS ist also nicht injektiv. Dies beweist bereits Abbildung 3.7 auf Seite 47. Es kommt sogar hinzu, dass es einige Konfigurationen gibt, für die diese Funktion nicht mal surjektiv ist.

Während der Interaktionen Panning und Resize werden naheliegenderweise die Berechnungen des FISHEYE–VIEWS auf den *Bildschirmkoordinaten* des Fokus aufgebaut. Es wird also eine Pseudo–Inverse der Berechnungsfunktion gesucht, die die Bildschirmposition des Fokus in eine Position im Ausgangsbild umrechnet, obwohl der FISHEYE–VIEW noch nicht berechnet wurde.[23] Abbildung 3.9 verdeutlicht die Vorgehensweise während des Panning und Resize.

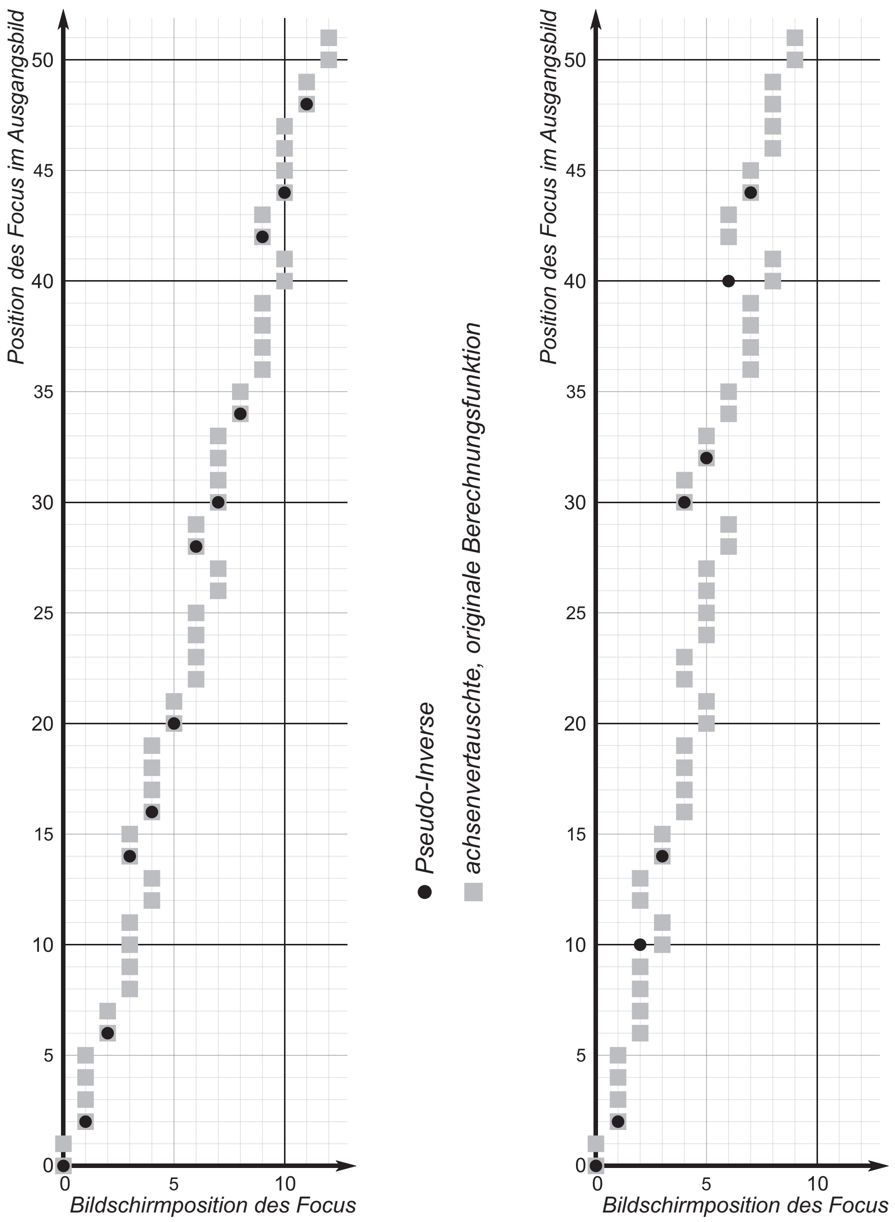
Das Ziel für die Bestimmung der Pseudo–Inversen besteht darin, dass vorgegebene und berechnete Bildschirmkoordinaten des Fokus identisch sind. Noch steht der mathematische Beweis über die Existenz bzw. Nichtexistenz einer sol-

[23]Für den Fall, dass der FISHEYE–VIEW schon berechnet wurde, ist es leicht, Bildschirmkoordinaten in Ausgangsbildkoordinaten umzurechnen. Dies geschieht z.B. bei der Jump-Interaktion.

chen Pseudo–Inversen für alle Konfigurationen aus. Aus diesem Grund wird die in der Implementation umgesetzte Pseudo–Inverse an dieser Stelle nicht weiter vorgestellt. Sie entspricht weitestgehend den Erwartungen, liefert aber für Konfigurationen mit sehr vielen Kontextringen eventuell Fehlstellen.[24] Diese äußern sich für den Nutzer durch ein „Kleben“ des Fokus beim Panning bzw. Resize.

Abbildung 3.10 zeigt zwei Plots der Pseudo–Inversen im Vergleich zur achsenvertauschten, originalen Berechnungsfunktion. Die Punkte der *perfekten* Pseudo–Inversen würden alle auf den Punkten der achsenvertauschten, originalen Funktion liegen. Probleme würden dann nur noch im Fall einer nicht-surjektiven Berechnungsfunktion auftreten. Das sogenannte „Kleben“ des Fokus beim Verschieben oder Vergrößern/Verkleinern ist Resultat der Fehlstellen, wie sie in Abbildung 3.10b an den Bildschirmpositionen 2 und 6 gezeigt werden. Wenn der Nutzer bei dieser Konfiguration z.B. den Fokus langsam um jeweils eine Bildschirmkoordinate nach links verschiebt, dann wird sich der Fokus nicht über die Bildschirmposition 2 hinaus weiter nach links verschieben lassen. Befindet er sich an Bildschirmposition 2, so berechnet die Pseudo–Inverse eine Position im Ausgangsbild von 10. Gibt man jedoch diesen Eingangswert in die Berechnungsfunktion für den FishEye–View ein, so erhält man eine Bildschirmposition von 3. Der Fokus springt also um einen Pixel zurück nach rechts. Verschiebt der Nutzer nun weiter, kommt er mit dem Fokus zunächst wieder auf die Bildschirmposition 2 und springt sofort zurück auf Position 3. Dieser „Falle“ kann man nur entkommen, wenn man den Fokus mit höherer Geschwindigkeit (also um 2 oder mehr Bildschirmkoordinaten in einem Schritt) verschiebt.

[24]Vorgegebene Bildschirmkoordinate des Fokus, die nicht mit der berechneten Koordinate übereinstimmt.



(a) Grundauflösungen 1, 2, 4 und 8. Kein Zoom. Keine Fehlstellen.

(b) Grundauflösungen 1, 2, 4, 8 und 16. Kein Zoom. Fehlstellen im gezeigten Ausschnitt bei Bildschirmpositionen 2 und 6.

Abbildung 3.10: Funktionsplots der Pseudo–Inversen im Vergleich zur achsenvertauschten, originalen Berechnungsfunktion. Die Angaben beziehen sich auf die linke Koordinate des Fokus.

3.5.3 Interaktivitätsmodell

Abbildung 3.11 zeigt das Interaktionsverhalten des FishEye–Views in Form eines Automaten.

Es fällt sicherlich auf, dass es drei verschiedene Darstellungsarten gibt: Schnelldarstellung, Qualitätsdarstellung und Waveletdarstellung. Diese Unterteilung ermöglicht es, sehr schnell den FishEye–View auf dem Bildschirm zu visualisieren und ihn anschließend zu verfeinern.

Zunächst sollen die ersten beiden Darstellungsarten erklärt werden. Beide beziehen ihre Daten aus einer lokalen Verwaltung für unkomprimierte Bilddaten (wie z.B. im Kapitel 2 vorgestellt). Die Schnelldarstellung benutzt jedoch Bilddaten geringerer Auflösung als eigentlich nötig wären, während die Qualitätsdarstellung höher aufgelöste Daten nutzt. Sei beispielsweise ein Rechteck mit der Auflösung $x : y$ darzustellen, wobei o.B.d.A. $x < y$ sei. Unter der Maßgabe, dass nur pixelquadratische Auflösungen vorliegen, gibt es zwei Möglichkeiten die Auflösung $x : y$ zu erzeugen:

1. **Laden der niedrigeren Auflösung** $x : x$**.** Die Auflösung $x : y$ wird dann mittels Streckung in Richtung der y-Achse erzeugt. Dieses Verfahren wird bei der Schnelldarstellung angewandt.
2. **Laden der höheren Auflösung** $y : y$**.** Die Auflösung $x : y$ wird dann mittels Stauchung in Richtung der x-Achse erzeugt. Dieses Verfahren wird bei der Qualitätsdarstellung angewandt.

Es ist leicht zu ersehen, dass Rechtecke mit Auflösungen der Form $x : y$ mit $x = y$ bereits in der Schnelldarstellung in voller Qualität dargestellt werden. Diese Regionen des Bildes müssen durch die Qualitätsdarstellung nicht erneut gezeichnet werden. Die Skalierung der Bilddaten erfolgt bei beiden Darstellungsarten aus Geschwindigkeitsgründen mit Hilfe der Windows-API, was gewisse Artefakte mit sich bringt.

Die Waveletdarstellung erzeugt die Bilddaten direkt aus den übertragenen Waveletkoeffizienten und ermöglicht somit eine exakte und hochqualitative Skalierung einzelner Bildbereiche.

Um nicht zu große Wartezeiten durch die Qualitäts- bzw. Waveletdarstellung zu erzeugen, sind diese nach Abbarbeitung jedes Teilrechtecks unterbrechbar.[25] Die Unterbrechung kann mittels einer Interaktion erfolgen. In diesem Fall wird vom Idle-Zustand sofort in den entsprechenden Interaktionszustand gesprungen.

Wie bereits bekannt, benötigen die Interaktionen Panning und Resize ein Neuzeichnen des FishEye–Views. Diese Aktionsketten werden in Abbildung 3.11 mittels gepunkteter bzw. gestrichelter Linien dargestellt. Sobald neue Parameter durch die Interaktion vorliegen, erfolgt eine Neuberechnung und Schnelldarstellung. Während der Interaktion ist die Schnelldarstellung unterbrechbar, falls weitere Eingaben vorliegen.

Die einzelnen Aktionen sind abbrechbar (in der Abbildung 3.11 mit „ESC“ für ESC-Taste bezeichnet). Der FishEye–View geht in den Zustand vor Beginn der Interaktion zurück.

[25] Für die Schnelldarstellung gilt dies nicht, da zumindest eine gültige Darstellung des FishEye–Views auf dem Bildschirm gezeigt werden soll.

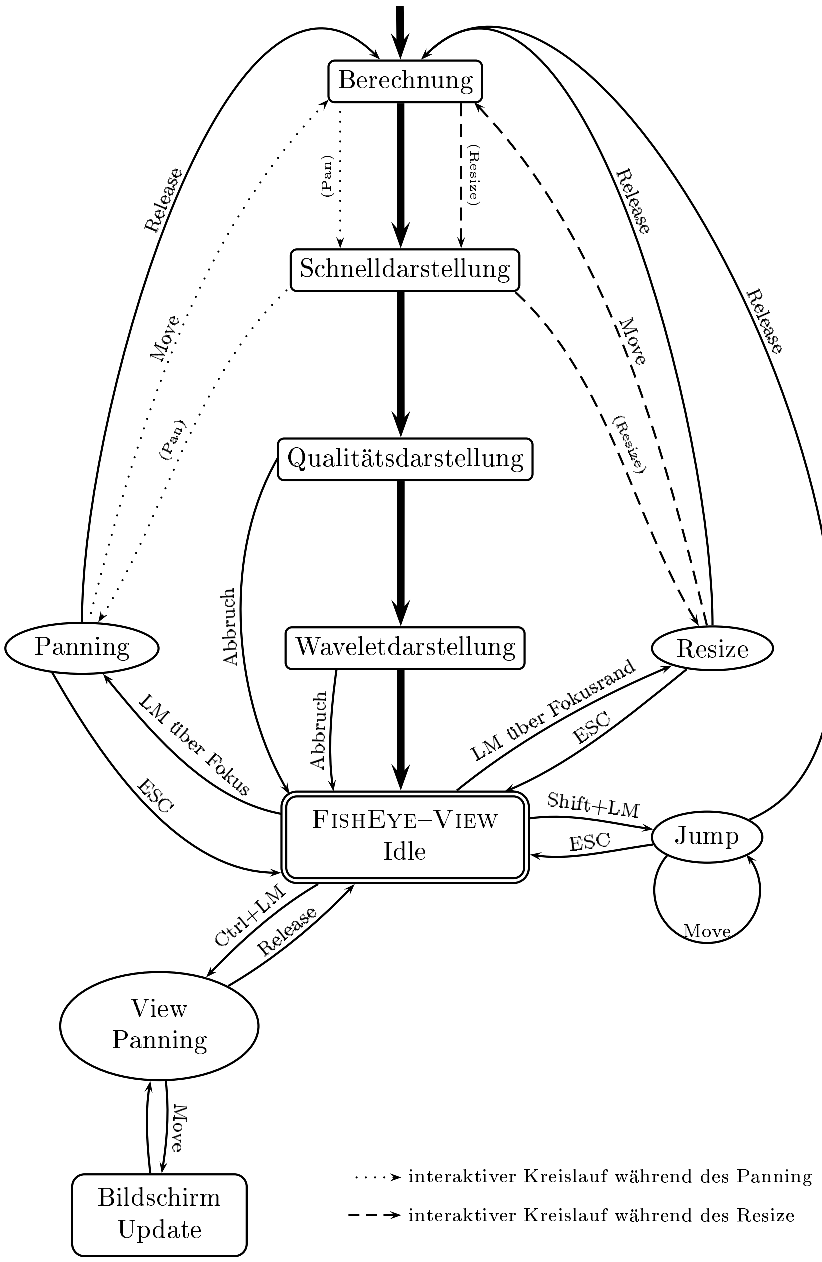


Abbildung 3.11: Das Interaktivitätsmodell des monofokalen rechteckigen FishEye–Views. „LM“ bedeutet, dass die linke Maustaste (evt. im Zusammenhang mit der „Shift“- bzw. „CTRL“-Taste) gedrückt wird. „Move“ bedeutet, dass die Maus bei gedrückter Tastenkombination bewegt wird. „Release“ gibt den Moment des Loslassens der Tasten an.

3.6 Der multifokale rechteckige FishEye–View

3.6.1 Motivation

Wenn man längere Zeit mit dem monofokalen FishEye–View arbeitet, stellt sich die Frage, ob es nicht auch möglich sei, einen FishEye–View mit mehreren Fokusregionen zu erzeugen. Dies könnte beispielsweise dann von Interesse sein, wenn der Betrachter verschiedene Ausschnitte des Ausgangsbildes im Detail miteinander vergleichen will. Eine weitere Anwendungsmöglichkeit ergibt sich bei einer Online–Konferenz. Die den Betrachter besonders interessierenden Konferenzteilnehmer werden innerhalb einer Fokusregion dargestellt, während die anderen nur in den Kontextringen abgebildet werden. Dies spart Bandbreite, was besonders bei Online–Konferenzen sehr wichtig ist. Eine Abbildung, die dieses Szenario nachstellen soll, findet sich im Anhang C auf Seite 80.

Sarkar und Reiss stellten bereits in [SR92] ein allgemeines Verfahren für einen FishEye–View mit mehreren Foki vor. Jedoch basiert ihr Ansatz auf grafischen Layouts und unterstützt nicht mehrere Kontextringe. Zudem wurde nicht die Frage der Überlappung von Foki beantwortet.

In diesem Abschnitt soll ein Algorithmus vorgestellt werden, der den multifokalen FishEye–View auf der Basis des monofokalen berechnet. Dabei können alle Vorteile der vorhandenen Technik eingesetzt werden.

3.6.2 Vom monofokalen zum multifokalen FishEye–View

Bei dem monofokalen FishEye–View fällt die Auflösung der dargestellten Bildinformation mit zunehmendem Abstand von der Fokusregion stufenweise bis zum Bildrand hin ab. In dieser Art und Weise muss auch jede Fokusregion im multifokalen FishEye–View kontextiert werden. Jedoch werden die Kontextringe einer Fokusregion A durch einen anderen Fokus B überlagert. Es ist nicht möglich, die Kontextringe von A und den Fokus B an der gleichen Stelle darzustellen, da unterschiedliche Auflösungen und nicht vereinbare Geometrien vorliegen.

Also muss die Lösung darin gesucht werden, Fokus A nur soweit zu kontextieren, dass die Darstellung von Fokus B mit seinen Kontextringen nicht behindert wird. Es ergibt sich eine Aufteilung des Ausgangsbildes in verschiedene Einflußzonen. Der Fokus befindet sich in der (wie auch immer gearteten) „Mitte" seiner Einflußzone und seine Kontextringe fallen in ihrer Auflösung zum Rand der Zone hin ab. Den Rand der Zone bilden der Bildrand und Ränder anderer Einflußzonen. Grenzt eine Seite an eine andere Zone, so steigt die Auflösung wieder stufenweise an, bis der andere Fokus erreicht ist usw.

Beeinflussen sich die Fokusregionen nicht gegenseitig durch Überlappung, so ist es ohne weiteres möglich, das multifokal darzustellende Ausgangsbild in Teilbilder zu zerteilen und diese einzeln mit dem monofokalen FishEye–View darzustellen. Fügt man die Ergebnisse wieder zu einem Bild zusammen, erhält man eine erste Vorstellung von einem multifokalen FishEye–View, wie in Abbildung 3.12 dargestellt. Dabei muss die Einteilung des Ausgangsbildes nicht immer einheitlich erfolgen, wie Abbildung 3.13 verdeutlicht. Weiterhin können zwei Fokusregionen im Zusammenspiel zwei weitere Foki erzeugen. Die Beeinflussung der Foki untereinander und die Auswirkung auf die Darstellung ist

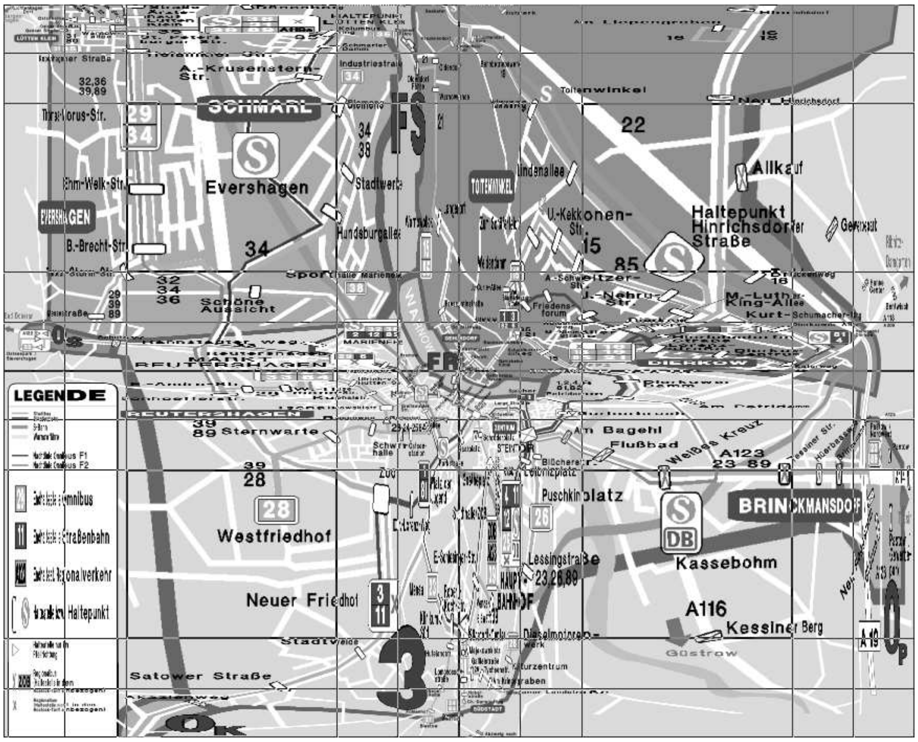
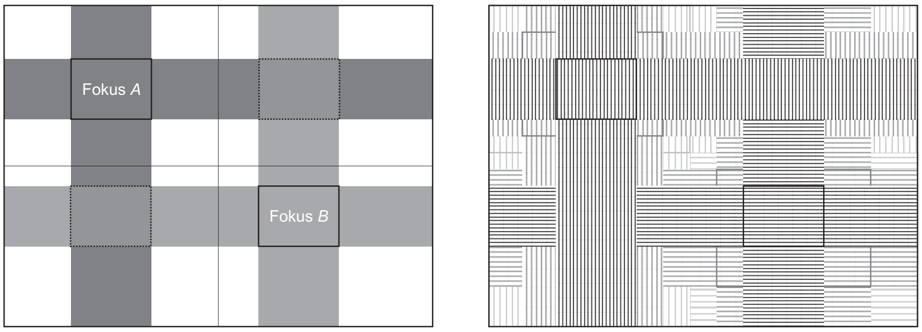


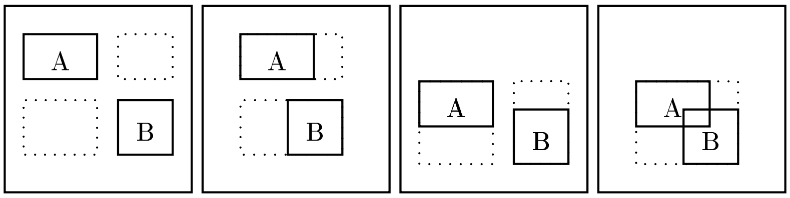
Abbildung 3.12: Gleichmäßiger multifokaler FISHEYE–VIEW mit augenscheinlich 4 Fokusregionen. Im weiteren wird erkennbar werden, dass die Abbildung auch mit 2, 3 oder mehr als 4 Fokusregionen erstellt worden sein kann.



(a) Das Entstehen neuer Foki. Die beiden senkrechten Linien zeigen die Mitte des Bildes.

(b) Die mögliche Komplexität multifokaler FISHEYE–VIEWS.

Abbildung 3.13: Ungleichmäßiger multifokaler FISHEYE–VIEW.



(a) Keine Überlappung in x- oder y-Richtung.

(b) Überlappung nur in x-Richtung.

(c) Überlappung nur in y-Richtung.

(d) Überlappung in x- und in y-Richtung.

Abbildung 3.14: Mögliche gegenseitige Beeinflussungen zweier Foki.

Thema des nächsten Abschnitts.

3.6.3 Beeinflussung der Fokusregionen

Wie bereits Abbildung 3.13a zeigt, können sich Fokusregionen derart beeinflussen, dass neue Foki zwangsweise hinzukommen.

Ausgangspunkt für die folgenden Betrachtungen seien Fokus A und Fokus B mit den Koordinaten $[a_L, a_T, a_R, a_B]$ bzw. $[b_L, b_T, b_R, b_B]$. Folgende Fälle gegenseitiger Beeinflussung sind zu unterscheiden (Abbildung 3.14):

1. **Keine Überlappung in x- oder y-Richtung.**
   Es bilden sich zwangsweise zwei implizite Fokusregionen an den Stellen $[b_L, a_T, b_R, a_B]$ und $[a_L, b_T, a_R, b_B]$.

2. **Überlappung nur in x-Richtung.**
   Beide Fokusregionen werden in x-Richtung erweitert, so dass sie sich vollständig in dieser Richtung überlappen. Für die Fokusregion A ergibt das die Koordinaten $[\min(a_L, b_L), a_T, \max(a_R, b_R), a_B]$ und für den Fokus B $[\min(a_L, b_L), b_T, \max(a_R, b_R), b_B]$.

3. **Überlappung nur in y-Richtung.**
   Analog zur Überlappung nur in x-Richtung.

4. **Überlappung in x- und in y-Richtung.**
   Es bildet sich aus beiden Foki *eine* neue Fokusregion mit den Koordinaten $[\min(a_L, b_L), \min(a_T, b_T), \max(a_R, b_R), \max(b_B, b_B)]$.

Explizite Fokusregionen sind die vom Nutzer gewünschten Foki, während Gesamt–Fokusregionen alle die Teile des Ausgangsbildes sind, die die Fokus-Auflösung besitzen. Damit gehören auch die expliziten zu den Gesamt–Fokusregionen. Implizite Fokusregionen sind die Gesamt–Fokusregionen, die nicht explizit vom Nutzer bestimmt wurden. Im weiteren wird mit „Fokus“ bzw. „Foki“ von den Gesamt–Fokusregionen gesprochen, sofern nicht anders angegeben.

Es genügt, zwei explizite Foki und deren gegenseitige Beeinflussung zu untersuchen. Fälle mit mehr als zwei expliziten Foki lassen sich darauf reduzieren, indem alle möglichen Paare einzeln betrachtet werden. Die Reihenfolge ist dabei unerheblich.

3.6.4 Bestimmung der Einflußzonen

Hat man nun eine Menge von Foki im Ausgangsbild auf diese Art und Weise verteilt, so müssen ihre Einflußzonen berechnet werden. Die Zone eines Fokus darf nicht einen anderen Fokus beinhalten, sonst wäre die Kontextierung nicht möglich. Die Grenzen der Einflußzonen müssen also zwischen den Foki verlaufen.

Diesem Grundsatz folgend müssen für jeden Fokus die Nachbarn in alle vier Richtungen (nach links, rechts, oben und unten) bestimmt werden. Dies erfolgt für einen Fokus A mit den Koordinaten $[a_L, a_T, a_R, a_B]$ so, dass aus allen anderen Foki B^i diejenigen Foki B' mit $[b'_L, b'_T, b'_R, b'_B]$ herausgesucht werden, für die eine der folgenden Bedingungen gilt:

$$a_L - b'_R = \min(a_L - b^i_R) \geq 0 \tag{3.19a}$$
$$a_T - b'_B = \min(a_T - b^i_B) \geq 0 \tag{3.19b}$$
$$b'_L - a_R = \min(b^i_L - a_R) \geq 0 \tag{3.19c}$$
$$b'_B - a_T = \min(b^i_B - a_T) \geq 0 \tag{3.19d}$$

Die Einflußzone des Fokus A ergibt sich nunmehr aus den in (3.19) berechneten Abständen zu den Nachbarn. Die simpelste Methode der Bestimmung wäre dann gegeben, wenn alle Foki gleich behandelt werden. In diesem Fall verläuft die Grenze genau in der Mitte zwischen A und seinen Nachbarn. Gibt es für eine Richtung keinen Nachbarn, dann ist der Bildrand die Grenze.

Problematischer fällt die Berechnung aus, wenn man an die expliziten Foki zusätzlich noch Prioritäten vergibt. Durch die Beeinflussung der Fokusregionen müssen auch die impliziten Foki die Prioritäten in bestimmter Weise übernehmen. Die neuen bzw. erweiterten Foki „erben" die Prioritäten ihrer Erzeuger an den übernommenen Koordinaten. Also hat ein Fokus für jede Richtung eine Priorität. Bereits Abbildung 3.13a zeigt das, denn wenn die beiden expliziten Foki unterschiedliche Prioritäten haben, können diese durch die neuen impliziten Foki nicht einfach übernommen werden. Jeder Fokus muss mit seinen horizontalen Partnern eine gleiche Priorität nach oben bzw. unten haben und für die vertikalen Partner gilt diese Prioritätengemeinschaft für die linke bzw. rechte Seite ebenso. Es ist leicht ersichtlich, dass nur unter dieser Maßgabe die Kontextierung zwischen allen Partnern in die jeweilige Richtung auf die gleiche Weise erfolgen kann. Dieses Prinzip gilt allgemein für die Kontextierung der Foki und hat Auswirkungen auf die Wahl und Vererbung der Kontextauflösungen, Sichtbarkeiten und Größenverhältnisse von Kontextringen. Diese Parameter müssen für alle Fokuspartner in die jeweilige Richtung identisch sein, können jedoch für einen Fokus in alle vier Richtungen unterschiedlich sein.

Damit klärt sich auch die Frage, ob sich die Foki in unterschiedlichen Zoomstufen befinden können. Die Antwort ist nein, denn die gleiche Kontextierung der horizontalen bzw. vertikalen Partner wäre dann nicht gegeben. Zumindest muss man diese Möglichkeit für den hier vorgestellten Ansatz verneinen. Sehr wohl ist es aber möglich, in den *gesamten* multifokalen FISHEYE–VIEW zu zoomen, also alle Foki befinden sich in derselben Zoomstufe.

3.6.5 Algorithmus zur Berechnung und Darstellung des multifokalen FishEye–Views

Der in diesem Abschnitt vorgestellte Algorithmus bezieht sich teilweise auf vorangegange Abschnitte, in denen die mathematischen Grundlagen gelegt wurden. Es sollen an dieser Stelle nicht noch einmal alle Formeln aufgelistet, sondern das Augenmerk auf die grundlegende Vorgehensweise gelenkt werden.

1. $\mathbf{A}$ sei die Menge aller expliziten Fokusregionen, $\mathbf{B}$ sei die Menge aller Gesamt-Fokusregionen. $n = \|\mathbf{A}\|$. $m = \|\mathbf{B}\| = \emptyset$.
2. Für alle $a_i \in \mathbf{A}$, $i = (1, \ldots, n-1)$
   Für alle $a_j \in \mathbf{A}$, $j = (i+1, \ldots, n)$
   (a) Überlappen sich a_i und a_j in mindestens eine Richtung, so erweitere die Regionen dementsprechend und füge beide in $\mathbf{B}$ ein.
   (b) Überlappen sich a_i und a_j nicht, so bilde 2 implizite Fokusregionen und füge diese in $\mathbf{B}$ ein. Füge a_i in $\mathbf{B}$ ein.
3. Eliminiere redundante Elemente in $\mathbf{B}$.
4. Berechne die Einflußzone jeder Fokusregion in $\mathbf{B}$.
5. Berechne für jede Fokusregion in $\mathbf{B}$ den monofokalen FISHEYE–VIEW.
6. Fasse benachbarte Rechtecke mit gleicher Auflösung zusammen.
7. Stelle alle Rechtecke dar.

Kapitel 4

Kommunikation mit dem Bildserver

Dieses Kapitel widmet sich der Kommunikation zwischen dem Bildserver und dem Bildclient. In den vorangegangenen Kapiteln wurden bereits die Techniken vorgestellt, die auf dem Client für die Verwaltung der Bilddaten und deren Anzeige sorgen. Somit bleibt noch zu klären, wie die Daten vom Bildserver angefordert werden und wie die Prozesse auf dem Server selbst ablaufen.

4.1 Protokoll

Primäres Ziel der Datenübertragung und Kommunikation zwischen Bildserver und –client ist Redundanzfreiheit, da die Ressourcen zur Übertragung (insbesondere im mobilen Kontext) gering bemessen sind. Somit ist es notwendig, dass sowohl Server als auch Client jederzeit Wissen darüber verfügen, welche Bildteile in welcher Detailstufe bereits übertragen worden sind.

Unterschiedliche Anzeigetechniken erfordern verschiedene Vorgehensweisen für die Aktualisierung der Spezifikation zur Datenanforderung. So ist es bei der Anwendung des in Kapitel 3 vorgestellten FishEye–Views sinnvoll, nach einer Änderung der Ansicht durch den Nutzer, die bestehende Übertragung abzubrechen und eine Datenanforderung an den Server zu stellen, die der neuen Ansicht entspricht. Andere Anzeigetechniken gehen jedoch von dem Ansatz aus, zu einer bestehenden Anforderung weitere Regions of Interest *hinzuzufügen*.

Diese Problematiken erfordern eine dauerhafte Verbindung zwischen Server und Client. Da es jedoch mittels des HTTP–Protokolls nicht möglich ist, eine dauerhafte Verbindung zu etablieren, muss die Kommunikation über sogenannte „virtuelle HTTP–Verbindungen“ erfolgen. Eine virtuelle Verbindung ist durch eine Verbindungs–ID definiert, die beim ersten Aufruf des Servers durch den Client eindeutig durch den Server vergeben wird. Diese ID muss dem Client in der auf den ersten Request folgenden Antwort übermittelt werden, so dass alle weiteren Anfragen des Clients an den Server mit dieser ID gekenzeichnet und somit eindeutig der virtuellen Verbindung zugeschrieben werden können. Nur so kann das serverseitige Wissen über den letzten Stand der Übertragung auch über die Lebensdauer einer HTTP–Verbindung hinaus genutzt werden.

Die Spezifikation für die Etablierung einer virtuellen Verbindung beim ersten Aufruf des Servers sieht somit wie folgt aus:[1]

```
 CGI-Input::  PIC=[Bildname]&
              IRL=[Anzahl der unabhängigen Auflösungsstufen]
CGI-Output::  <ID>[ID]</ID><SIZEX>[Bildgröße X]</SIZEX>
              <SIZEY>[Bildgröße Y]</SIZEY>
```

Die ID, die Bildparameter und die Anzahl der unabhängigen Auflösungsstufen sind Werte größer 0. Für den Fall, dass der Server wegen Überlastung oder sonstigem keine virtuelle Verbindung aufbauen kann, schickt er dem Client nur eine Fehlermitteilung in Form einer Textnachricht:

```
Fehler:
CGI-Output::  <ERROR>[Fehlertext]</ERROR>
```

Ließ sich eine virtuelle Verbindung aufbauen, hat der Client nunmehr die Möglichkeit, anhand der Bildgröße die erforderlichen Berechnungen zur Darstellung der Bilddaten vorzunehmen. Dies ergibt (je nach Darstellungstechnik) eine Reihe von Regions of Interest, für die der Client beim Server Bilddaten anfordern muss. Dies geschieht mittels der durch Uwe Rauschenbach spezifizierten Syntax, die dem Anhang D zu entnehmen ist. Die Anforderung von RoI–Daten wird durch folgende Spezifikation erfüllt:

```
 CGI-Input::  ID=[ID]&CMD1=[RoI-Daten]&...&CMDn=[RoI-Daten]&
              Append=[Y/N]
CGI-Output::  <DATA>[komprimierte Bilddaten]</DATA>
```

Das schliessende </DATA>–Tag ist optional, d.h. die zuvor übertragenen komprimierten Bilddaten werden auch ohne den Erhalt dieses Tags interpretiert. Dies ist für den Fall notwendig, dass die Übertragung unterbrochen wurde.

Alle weiteren Anforderungen des Clients werden in gleicher Weise durchgeführt. Jedoch muss hierbei angegeben werden, ob die eventuell noch bestehende Übertragung abgebrochen oder die neue Spezifikation in die bisherige integriert werden soll. Dazu dient die Angabe „Append“.

Im allgemeinen wird der Client bei seiner Terminierung nicht mehr in der Lage sein, dem Server eine Nachricht über die Beendigung der virtuellen Verbindung zu schicken. Somit wird für diesen Fall auch keine Protokollspezifikation angegeben. Die Verbindungsdaten auf dem Server werden nach einer angemessenen Zeit gelöscht.

[1] Die in eckigen Klammern gesetzten Zeichen werden durch den bezeichneten Inhalt ersetzt. Die durch „<“ und „>“ begrenzten Zeichen gehören inklusive der Begrenzungszeichen zum Datenstrom. Bei der Angabe der [Anzahl der unabhängigen Auflösungsstufen] handelt es sich um einen für die Initialisierung der Waveletkomprimierung wichtigen Wert.

4.2 Serverseitiger Prozessablauf

Der Bildserver muss neben der Abarbeitung des Kommunikationsprotokolls die folgenden Aufgaben erfüllen:

- Generierung eindeutiger IDs für virtuelle HTTP–Verbindungen.
- Komprimierung der Bilddaten.
- Re–Initialisierung einer Übertragung.

Die Komprimierung der Bilddaten erfolgt mittels einer durch Uwe Rauschenbach implementierten Bibliothek und ist nicht Gegenstand der Betrachtungen in dieser Arbeit.

Zur Generierung eindeutiger IDs und zur Re-Initialisierung einer Übertragung sind einige Überlegungen notwendig, denn durch jede Anfrage des Clients wird beim Server ein neuer Bildserverprozess gestartet, der zunächst keine Informationen über bestehende Übertragungen oder bereits vergebene IDs hat. Somit muss zwischen den verschiedenen Prozessen eine Kommunikation erfolgen, die darüber Aufschluss gibt.

Da die Implementation auf allen Plattformen lauffähig sein soll, ist eine betriebssystemabhängige Prozesskommunikation auszuschliessen. Somit bietet sich eine datenträgergestützte Kommunikation an, die darauf basiert, dass die notwendigen Informationen in Dateien und Verzeichnisstrukturen abgelegt werden. Dieses Medium erlaubt dann keine gleichzeitigen Zugriffe verschiedener, konkurrierender Prozesse, wenn die jeweiligen Dateien durch einen der konkurrierenden Prozesse gesperrt werden. Das Betriebssystem erfüllt somit die Überwachung der Konkurrenz.

Die folgenden Abschnitte behandeln die Strategien zur Kommunikation.

4.2.1 Strategie zur Vergabe eindeutiger IDs

In einem separaten Verzeichnis werden Unterverzeichnisse angelegt, deren Namen nur aus einer Zahl (> 0) bestehen. Soll eine neue virtuelle Verbindung etabliert werden, so werden die Namen der Unterverzeichnisse in diesem Verzeichnis entsprechend ihrer numerischen Ordnung eingelesen. Die kleinste, noch nicht in dieser Liste vorkommende Zahl wird an die neue virtuelle Verbindung als ID vergeben.

In den für jede virtuelle Verbindung angelegten Verzeichnissen werden alle notwendigen Daten zum Erhalt, zur Verwaltung und zur Protokollierung der virtuellen Verbindung angelegt. Diese Verzeichnisse werden ID–Verzeichnisse genannt.

Da alle Dateien (und somit auch Verzeichnisse) in jedem Betriebssystem mit einem Datum und einer Zeit versehen werden, zu der sie angelegt wurden, ist es für einen speziellen, nicht durch den HTTP–Server gestarteten Prozess sehr einfach, die Daten aller virtuellen Verbindungen, die älter als ein bestimmtes Datum sind, zu löschen. Somit wird garantiert, dass der Datenträger genügend freien Speicherplatz für weitere Verbindungen und andere Aufgaben hat, die Zahlenwerte der vergebenen IDs nicht zu groß werden und das Einlesen der Dateinamen nicht zuviel Zeit in Anspruch nimmt.

4.2.2 Strategie zum Abbruch von Übertragungen

Um das serverseitige Wissen über den Zustand der Datenübertragung auch nach der Beendigung der letzten HTTP–Verbindung zu erhalten, muss bei der Terminierung des Bildserverprozesses der aktuelle Zustand in einer Datei im ID–Verzeichniss festgehalten werden. Im Falle einer erneuten Anforderung durch den Client müssen die Informationen dieser Datei eingelesen und berücksichtigt werden. Der Name dieser Zustands–Datei wird wie folgt festgelegt:

```
state
```

Fordert der Client aufgrund einer Nutzerinteraktion o.ä. eine neue Menge von RoIs an, obwohl zu diesem Zeitpunkt noch Daten aus einer vorherigen Anforderung übertragen werden, so ist es nicht notwendig den Inhalt dieser Datei auszuwerten, da weiterhin ein Bildserverprozess existiert, der über den Zustand der Übertragung verfügt. Die Existenz dieses aktiven Prozesses wird im ID–Verzeichnis durch folgende (leere) Aktiv–Datei angezeigt:[2]

```
active
```

Durch die neue Anforderung wird auf dem Server jedoch ein neuer Prozess gestartet, der dem bereits aktiven Prozess das Vorhandensein einer neuen Spezifikation zur Datenübertragung anzeigen muss. Dazu schreibt der neue Prozess diese Spezifikation in die folgende Warte–Datei:

```
wait
```

Wie bereits erwähnt, ist es für einige Anzeigetechniken sinnvoll, diese Spezifikation in die bestehende Datenübertragung zu integrieren, während bei anderen Anwendungen die neue Anforderung die bisherige ersetzen soll. Dem neu aufgerufenen Prozess wird diese Information über das bereits beschriebene Protokoll mitgeteilt.

Im Falle einer Integration in die bestehende Spezifikation, wird die neue Anforderung an den Inhalt der Warte–Datei mit dem Vermerk zur Integration angefügt. Falls die neue Spezifikation jedoch die bestehende ersetzen soll, so wird auch der Inhalt der Warte–Datei durch die neue Anforderung ersetzt. An dieser Stelle ist die Aufgabe des neuen Bildserverprozesses beendet.

Der aktive Prozess muss während seiner Ausführung in regelmässigen Abständen das Vorhandensein und den Inhalt der Warte–Datei überprüfen. Je nach dem Vermerk zur Integration in oder Ersetzung der bisherigen Spezifikation wird der Inhalt der Warte–Datei ausgewertet und die laufende Datenübertragung angepasst.

Es ist zu beachten, dass nach der Auswertung der Warte–Datei diese durch den aktiven Prozess gelöscht wird. Weiterhin kann es vorkommen, dass mehrere neue Prozesse den Inhalt der Warte–Datei verändern. In diesem Fall befinden

[2] Bei seiner Beendigung löscht der aktive Prozess diese Datei.

sich (falls Integration gewünscht ist) mehrere Spezifikationen in der Warte–Datei.

Der Vorteil dieser Vorgehensweise besteht darin, dass sie auf allen Systemen zum Einsatz kommen kann. Zwar ist für die Verwaltung und den Ausschluss der Konkurrenz ein gewisser Overhead nötig, jedoch ist dieser vergleichsweise gering mit der zur Komprimierung der Bilddaten benötigten Zeit.

Kapitel 5

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, Mechanismen zu entwickeln, die eine redundanzfreie Übertragung und platzsparende interaktive Darstellung großer Bilddateien ermöglichen bzw. unterstützen. Diese Techniken wurden auf die von Uwe Rauschenbach spezifizierte Übertragungs- und Komprimierungstechnik abgestimmt, sind jedoch teilweise auch unabhängig davon einsetzbar.

Es wurde eine lokale Bilddatenverwaltung als Bindeglied zwischen Übertragungs– und Anzeigetechnik entwickelt, die in der Lage ist, Bilddaten virtuell zu verwalten. Somit werden unvollständig übertragene Bilder ressourcensparend gespeichert. Es können beliebig viele Auflösungen eines Bildes verwaltet werden. Zudem wurde bewiesen, dass unter bestimmten Voraussetzungen die zur Einsparung von Rechenzeit notwendige Redundanz limitiert werden kann. Weiterhin wurden Zugriffsoptimierungen umgesetzt, die bei Anforderungen seitens der Anzeigetechnik einen hohen Durchsatz gewähren.

Mit dem rechteckigen FishEye–View wurde eine Anzeigetechnik vorgestellt, die eine platzsparende Darstellung großer Bilddateien ermöglicht. Dies geschieht durch einen Fokus–und–Kontext Ansatz. Es existieren verschiedene Methoden zur Anpassung der Darstellung. Dazu gehören u.a. beliebig viele Kontextringe, einstellbare Größenverhältnisse zwischen den Ringen und Zoom in die Darstellung. Durch die Berechnungsvorschrift des FishEye–Views werden Auflösungsabweichungen und unterschlagene Pixelzeilen bzw. –spalten bei der Skalierung von Bilddaten weitestgehend vermieden. Es wurde ein Modell entwickelt, mit dessen Hilfe die interaktive Exploration des Bildes durch den Betrachter entscheidend unterstützt wird. Zudem wurde ein theoretischer Ansatz vorgelegt, der die Berechnung eines FishEye–Views mit mehreren Fokusregionen auf die des monofokalen zurückführt.

Abschliessend wurde die praktische Umsetzung der Kommunikation zwischen einem Bildserver und Bildclient vorgestellt. Das Konzept der virtuellen HTTP–Verbindungen wurde erläutert. Zudem sind Strategien entwickelt worden, die dieses Konzept praktisch umsetzen.

Im Anschluss an diese Studienarbeit sind die hier vorgestellten Techniken mit der durch Uwe Rauschenbach entwickelten Übertragungstechnik zu kombinieren und auf praktische Anwendbarkeit zu testen. Weiterhin beschäftigt sich ein Gruppe von Psychologen mit Tests zur Usability des FishEye–Views. Die hieraus gewonnenen Erkenntnisse werden bei der weiteren Entwicklung beachtet.

Literaturverzeichnis

[Fur82] George W. Furnas, *The fisheye view: A new look at structured files*, Technical Memorandum 82-11221-22, Bell Laboratories, 1982.

[Fur86] George W. Furnas, *Generalized fisheye views*, Proceedings of the ACM SIGCHI '86 Conference on Human Factors in Computing Systems (1986), 16–23.

[Lee97] Ho John Lee, *Too many pixels, not enough bandwidth*, Web Techniques Online Magazine, 06/97, http://www.webreview.com/97/06/06/feature/.

[Lee96] Ho John Lee, *High-resolution graphics with flashpix and the psifi internet imaging protocol*, Web Techniques Online Magazine, 12/96, http://www.webtechniques.com/features/1996/12/lee/lee.shtml.

[Möl92] Thomas Möller, *Fast bitmap stretching*, Graphics Gems III (1992), 4–7, Academic Press, San Diego.

[Rau98] Uwe Rauschenbach, *Progressive image transmission using levels of detail and regions of interest*, Proc. CGIM'98 – IASTED Conference on Computer Graphics and Imaging (Halifax, Nova Scotia, Canada), June 1-4 1998.

[RS98] Uwe Rauschenbach and Heidrun Schumann, *Flexible embedded image communication using levels of detail and regions of interest*, Proc. IMC '98 – Interactive Applications of Mobile Computing (Rostock, Germany), November 24-25 1998.

[SB92] Manojit Sarkar and Marc H. Brown, *Graphical fisheye views of graphs*, Proceedings of the ACM SIGCHI '92 Conference on Human Factors in Computing Systems, 1992.

[SR92] Manojit Sarkar and Steven P. Reiss, *Manipulating screen space with* StretchTools*: Visualizing large structure on small screen*, Brown University, Department of Computer Science, Providence, Rhode Island 02912, 1992.

[SSTR93] Manojit Sarkar, Scott S. Snibbe, Oren Tversky, and Steven P. Reiss, *Stretching the rubber sheet: A metaphor for viewing large layouts on small screens*, Brown University, Department of Computer Science, Providence, Rhode Island 02912, 1993.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Dateiformate der lokalen Bilddatenverwaltung

Dateiformat der Cache Index Datei

Typ: `ascii`

Geschrieben durch:

- `CLocalPic::SaveCache(const char* argIndexName, const char* argSaveDir ...);`

Gelesen durch:

- `CLocalPic::CLocalPic(const char* argpIndexName);`

In der Cache Index Datei werden die Namen der Cache Dateien festgehalten, die zu einem Bild gehören. Jede dieser Cache Dateien enthält die Bilddaten einer Auflösungsstufe. Neben der Anzahl der Cache Dateien und deren Namen werden auch noch die Längen der Dateinamen festgehalten. Dies ist wichtig für die Allokierung der Strings beim Laden des Cache.

Zeile	0	1 + 2*i (0 <= i < N)	2 + 2*i (0 <= i < N)
Funktion	Anzahl der Cache Dateien, die von dieser Index Datei referenziert werden.	Länge des folgenden Strings	Dateiname der i.ten Cache Datei (String der Länge L)
Inhalt	N	L	(z.B.) s11.1

Zeile	2*N + 1
Funktion	Dateikennzeichnung; Copyright
Inhalt	Adaptive Image Transmission - Local Image Management Cache Index File - MoVi Project – University of Rostock, Department of Computer Science, Institute of Computer Graphics 1998

Dateiformat der Cache Dateien

Typ: `binary`

Geschrieben durch:

- **Metabildinformation:**
  `CLocalPic::CLocalPic(int argNumOfRes, int* argXRes,...);`
- **Offset Tabelle (angelegt):**
  `CLocalPic::CLocalPic(int argNumOfRes, int* argXRes,...);`
- **Offset Tabelle (geändert):**
  `CLocalPic::GetPatch(CResolutionFileHandle* argpResfh, int argRow,...);`
- **Bilddaten (angelegt):**
  `CLocalPic::GetPatch(CResolutionFileHandle* argpResfh, int argRow,...);`
- **Bilddaten (geändert):**
  `CLocalPic::SaveRect(CPicHandle* argpPic);`

Gelesen durch:

- **Metabildinformation:**
  `CLocalPic::CLocalPic(const char* argpIndexName);`
- **Offset Tabelle:**
  `CLocalPic::GetPatch(CResolutionFileHandle* argpResfh, int argRow,...);`
- **Bilddaten:**
  `CLocalPic::LoadRect(CPicHandle* argpPic, unsigned int NumOfPads = 0,...);`

In einer Cache Datei werden die Bilddaten einer Auflösungsstufe verwaltet. Die Cache Datei ist in drei wesentliche Bestandteile eingeteilt: METABILDINFORMATION, OFFSET TABELLE und BILDDATEN.

Bei der METABILDINFORMATION handelt es sich um Information über das Bild, z.B. Größe des Bildes, Auflösung in dieser Datei, Farbmodus usw. Diese Information wird besonders beim Wiedereinlesen des Caches benutzt.

Die OFFSET TABELLE enthält Verweise auf die Bilddaten im hinteren Teil der Datei. Ein Offset ist ein 4 Byte langer Wert, der eine Stelle in der Datei eindeutig durch die Anzahl der vorhergehenden Bytes identifiziert. Um an diese Stelle zu gelangen, muß eine `FSEEK`-Operation vom Beginn der Datei an (`SEEK_SET`) durchgeführt werden. Der Offset wird durch eine `FTELL`-Operation bestimmt.

Der Aufbau der OFFSET TABELLE folgt den Kriterien für die lokale Bilddatenverwaltung. Es besteht die Anforderung, dass nur Teile eines großen Bildes lokal vorhanden sind. Deswegen wird das Bild in Patches unterteilt und nur mit Daten belegte Patches werden abgespeichert. Jeder Offset in der OFFSET TABELLE entspricht einem Patch und verweist eben genau auf den Beginn der Bilddaten dieses Patches in der Datei. Wenn der Offset eines Patches `NULL` ist, dann ist dieses Patch in der lokalen Bilddatenverwaltung noch nicht mit Daten belegt und existiert nicht in der Datei. Es kann dann am Ende der Datei angehangen werden, und der Offset für dieses Patch würde dann auf diese Stelle verweisen. Während die eigentlichen Patches in der Datei zumeist verstreut liegen, sind ihre Offsets in der OFFSET TABELLE wohl geordnet. An ihrer Position in der OFFSET TABELLE erkennt man, welches Patch mit diesem Offset referenziert wird. Die Offsets sind in der Reihenfolge der Patches zeilenweise abgelegt. Die Position Pos des Offsets des Patches in Zeile Z und Spalte S läßt sich wie folgt berechnen (Numerierung beginnt mit 0):

$$Pos = [\text{Beginn der OFFSET TABELLE}] + 4 \cdot (S + Z \cdot [\text{Anzahl der Spalten}])$$

Die BILDDATEN werden (wie oben beschrieben) in Patches abgespeichert und durch die OFFSET TABELLE referenziert. Es ist wichtig zu wissen, dass ein neues Patch immer an das Ende der Datei angefügt wird. Ist nun so ein Versuch erfolglos, weil z.B. die Festplatte voll ist, dann kann es ohne weiteres passieren, dass nur ein Teil des Patches an die Datei angehangen werden konnte. In diesem Fall wird das Patch als nicht vorhanden gekennzeichnet (`NULL` in der OFFSET TABELLE). Die geschriebenen Daten am Ende der Datei werden nicht zurückgenommen, so dass beim erneuten Hinzufügen eines Patches (wenn z.B. wieder Platz auf der Festplatte ist) zwischen den letzten beiden Patches Datenmüll steht. Dies hat jedoch auf die Funktion der lokalen Bilddatenverwaltung keinen Einfluß.

Es ist darauf zu achten, dass die lokale Bilddatenverwaltung aufgrund dieses Datenaufbaus nur Information darüber hat, ob Daten in einem Patch vorhanden sind oder nicht, aber nicht darüber, *wieviel* Daten vorhanden sind. Es kann nicht entschieden werden, ob in einem Patch eventuell nur eine Pixelzeile Bildinformation trägt. Dieses Problem kann man lösen, indem man nur vollständige Patches abspeichert. Dies ist bei der Wavelettransformation sehr leicht möglich.

Offset Byte	**0 – 4**	**5 – 8**	**9 - 12**
Funktion	kurze Dateikennzeichnung Local Image Management File Version 1	Offset der Metabildinformation	Offset der Offset Tabelle
Inhalt	LIMF1	0xXXXXXXXX	0xYYYYYYYY

Anmerkung: Die Addressen für die METABILDINFORMATION und die OFFSET TABELLE müssen in jedem Resolutionfile an der gleichen Stelle sein. Dies mag zwischen einzelnen Implementationen variieren, aber in *einer* Implementation muss sich diese Information immer an der gleichen Stelle befinden. Diese Stelle wird sich nämlich in privaten Memberattributen einer `CLocalPic`-Instanz gemerkt.

Offset Byte	**13 - 0xXXXXXXXX-1**
Funktion	lange Dateikennzeichnung; Copyright
Inhalt	Adaptive Image Transmission - Local Image Management File - MoVi Project - University of Rostock, Department of Computer Science, Institute of Computer Graphics 1998

METABILDINFORMATION I

Offset Byte	0xXXXXXXXXX - +3	+4 - +7	+8 - +11	+12 - +15
Funktion	Gesamtbildweite in Pixel	Gesamtbildhöhe in Pixel	Patchweite in Pixel	Patchhöhe in Pixel
Inhalt	z.B.: 4000	z.B.: 3000	z.B.: 128	z.B.: 128

METABILDINFORMATION II

Offset Byte	+16 - +19	+20 - +23	+24 - +27
Funktion	X-Auflösung	Y-Auflösung	Bildart (Farbe, Grau...)
Inhalt	z.B.: 2	z.B.: 2	z.B.: 2 (für Farbe)

OFFSET TABELLE

Offset Byte	0xYYYYYYYY - +3	+4 - +7	...	+4*(SP+ZE*AnzSpalten) - +3
Funktion	Offset der Bilddaten des Patches links oben	Offset der Bilddaten des Patches rechts neben dem Patch links oben	...	Offset der Bilddaten des Patches in Zeile ZE und Spalte SP, wenn Nummerierung mit 0 beginnt. Ein Offset = 4 Byte.
Inhalt	0xZZZZZZZZ	NULL = keine Daten	...	...

BILDDATEN

Offset Byte	0xZZZZZZZZ - +128*128*3
Funktion	Bilddaten des Patches links oben, wenn 128*128 Patch in Farbe
Inhalt	0xXX, 0xYY, 0xZZ, ...

Anhang B

Quelltext des Testprogramms zur Zugriffsoptimierung (FSEEK, MemCpy)

```
///////////////////////////////////
//
// Testprogram for comparing
// FSEEK method with MemCpy method
//
// author: Tino Weinkauf
// date: 01.07.1998
//
///////////////////////////////////

#include <stdio.h>
#include <time.h>
#include <memory.h>
#include <stdlib.h>

void main()
{
   FILE *stream;
   FILE *FSEEKlog, *MemCpylog;
   int i,j,k;
   int Loops;
   int MaxLoops = 2000;
   unsigned char buffer[13000];
   unsigned char picbuffer[17000];
   unsigned char *pData,*pPicData;
   time_t start,ende;
   clock_t cstart,cende;

   if ((FSEEKlog = fopen("D:\\Projekte\\...\\FSEEK.log", "w")) == NULL)
      {
         printf("The FSEEKlog was not opened.\n");
         exit(1);
      }
   else
```

```
   {
      printf("The FSEEKlog was opened.\n");
      fprintf(FSEEKlog, "[");
   }

if ((MemCpylog = fopen("D:\\Projekte\\...\\MemCpy.log", "w")) == NULL)
   {
      printf("The MemCpylog was not opened.\n");
      exit(1);
   }
else
   {
      printf("The MemCpylog was opened.\n");
      fprintf(MemCpylog, "[");
   }

//Main Loop
for (Loops=50;Loops<=MaxLoops;Loops+=50)
{
   printf("\n------------------\nLoops: %d / %d\n------------------\n",
      Loops, MaxLoops);

   if ((stream  = fopen("D:\\Projekte\\...\\testdata.1", "r+b")) == NULL)
   {
      printf("\nThe file was not opened.");
      exit(1);
   }
   else
      printf("\nThe file was opened.");

   ////////////////////////////////////////////////////////////////////
   //Load with FSEEK method
   time(&start);
   cstart = clock();
   printf("\nStarting Load with FSEEK Method:\t%ld\t%d\n",start,(int)cstart);
   for (k=0;k<Loops;k++)
   {
      srand(k);
      for (j=0;j<100;j++)
      {
         fseek(stream,rand()*1400000/RAND_MAX,SEEK_SET);

         pPicData = picbuffer;
         for (i=0;i<64;i++)
         {
            fread(pPicData,sizeof(unsigned char),192,stream);
            pPicData += 193;
         }
      }
      rewind(stream);
   }
   cende = clock();
   time(&ende);
   printf("Ending Load with FSEEK Method:\t%ld\t%d\n",ende,(int)cende);
   printf("\t==>Difference:\t%ld\t\t%2.3f\n",
      ende-start,(double)(cende-cstart)/CLOCKS_PER_SEC);
   fprintf(FSEEKlog, "{%d., %2.3f}, ",
      Loops, (double)(cende-cstart)/CLOCKS_PER_SEC);
   //END OF Load with FSEEK method
   ////////////////////////////////////////////////////////////////////

   fclose(stream);
```

```
      if ((stream  = fopen("D:\\Projekte\\...\\testdata.2", "r+b")) == NULL)
      {
         printf("\nThe file was not opened.");
         exit(1);
      }
      else
         printf("\nThe file was opened.");

      //////////////////////////////////////////////////////////////////
      //Load with MemCpy method
      time(&start);
      cstart = clock();
      printf("\nStarting Load with MemCpy Method:\t%ld\t%d\n",start,(int)cstart);
      for (k=0;k<Loops;k++)
      {
         srand(k);
         for (j=0;j<100;j++)
         {
            fseek(stream,rand()*1400000/RAND_MAX,SEEK_SET);
            fread(buffer,sizeof(unsigned char),12288,stream);

            pData = buffer;
            pPicData = picbuffer;
            for (i=0;i<64;i++)
            {
               memcpy(pPicData,pData,192);
               pData += 192;
               pPicData += 193;
            }
         }
         rewind(stream);
      }
      cende = clock();
      time(&ende);
      printf("Ending Load with MemCpy Method:\t%ld\t%d\n",ende,(int)cende);
      printf("\t==>Difference:\t%ld\t\t%2.3f\n",
         ende-start,(double)(cende-cstart)/CLOCKS_PER_SEC);
      fprintf(MemCpylog, "{%d., %2.3f}, ",
         Loops, (double)(cende-cstart)/CLOCKS_PER_SEC);
      //END OF Load with MemCpy method
      //////////////////////////////////////////////////////////////////

      fclose(stream);
   }

   fprintf(FSEEKlog, "]");
   fprintf(MemCpylog, "]");

   fclose(FSEEKlog);
   fclose(MemCpylog);
}
```

Anhang C

Multifokaler FishEye–View zum Online–Konferenz Szenario

Anhang D

Spezifikation der RoI–Syntax

Autor: Uwe Rauschenbach

```
RoI ASCII File format V 1.9

ZWECK
=====
Dieses Format dient zur Speicherung von RoIs (Regions
of Interest) für das Autorenwerkzeug, zum Datenaustausch
und als Eingabefile für den serverseitigen Bildkompressor.

Für den Dekompressor muß die Information in den Datenstrom
eingebettet werden. Zunächst wird aber ebenfalls das
ASCII-File als Zwischenlösung benutzt.


HISTORY
=======
1.1   erste freigegebene Version

1.2   PRIO <res> <prec> statt PRIO <xres> <yres> <prec>

1.3   PRIO <prec> statt PRIO <res> <prec>
      Index der Reference-RoI aus Header entfernt

1.4 Enthält zusätzlich padding-Informationen - obsolete

1.5 Angepaßt an neues LoD-Modell

1.6 Enthält zusätzlich Wavelet-Dekompositionsparameter

1.7 Enthält zusätzlich Wavelet-Filternummer

1.8 Enthält am Ende Liste von Autorenkommandos
    RoI-Spec am Dateiebeginn ist andere Notation für Autorenkommandos
   mit step==0

1.9 Redundanz entfernt: Bildgröße wird jetzt aus Bild gelesen


SYNTAX
```

```
======
<File> :: <Header> <RoI_Spec> {<RoI_Spec>} <number_of_auth_cmds> {<AuthorCmd> }

<Header> ::
"ROI19"  <EOL>
<path_and_name_of_image_file> <EOL>
<number_of_levels> <number_of_ir_levels> <filter_number> <EOL>
<number_of_RoIs> <EOL>

<AuthorCmd> ::
<step_cnt> <keyword> <params> <EOL>

<RoI_Spec> ::
@ NAME <name> <EOL>
@ LLOD <x_res> <y_res> <precision> <color><EOL>
@ PRIO <RoI_priority>
@ APRIO <x_resolution_prio> <x_resolution_prio> <precision_prio> <color_prio> <EOL>
<Footprint_spec>

<Footprint_spec> :: <Polygon> | <Rectangle> | <Ellipse>

<Polygon> ::
@ POLY <number_of_vertices> <EOL>
<X1> <Y1> <EOL>
...
<Xn> <Yn>  <EOL>

<Rectangle> ::
@ RECT <X1> <Y1> <X2> <Y2>  <EOL>

<Ellipse> ::
@ ELLIPSE <X_center> <Y_center> <dx> <dy> <EOL>


BEMERKUNGEN
===========
Alle Elemente werden durch Whitespace (Leerzeichen, TAB, Zeilenende)
getrennt. Leerzeilen sind nicht erlaubt.

ROI19 kennzeichnet die Version.

Kommentarzeilen beginnen mit # in der ersten Spalte.
Sie dürfen nicht im Header und nicht innerhalb eines <Polygon>
auftreten.

Attributzeilen beginnen mit @ in der ersten Spalte, gefolgt von
Whitespace, einem Schlüsselwort und Parametern. Jede Attributzeile
muß mit einen Zeilenendezeichen abgeschlossen werden.

X- und Y-Koordinaten sind Integer-Werte.
0<=x<image_width und 0<=y<image_height.

<path_and_name_of_image_file> gibt Pfad und Namen der Bilddatei an.
Der Pfad darf relativ sein; in diesem Fall muß beim Start des Decoders
gesichert werden, daß das richtige Arbeitsverzeihnis eingestellt ist.
Die Bilddatei muß existieren.

<number_of_levels> gibt die Anzahl der Wavelet-Zerlegungsstufen an
(>2). <number_of_ir_levels> gibt die Anzahl der unabhängigen
Zerlegungsstufen an (>=1; 1 entspricht klassischer Wavelet-Zerlegung).
<filter_number> gibt den Code des zu nutzenden Wavelet-Filers an
(analog zum Datenstrom-Header von MVWCodec). Default-Wert ist
```

8 (WT_HYB_53HAAR).

Die Attribute @ NAME, @ LLOD, @ PRIO @ APRIO sind OPTIONAL.
Wird ein optionales Attribut angegeben, so sind seine vollständigen
Argumente anzugeben.

Genau eine <Footprint_spec> je RoI ist VORGESCHRIEBEN.

@ NAME gibt der RoI einen Namen, unter dem sie im Autorensystem ansprechbar
ist. Es ist auch denkbar, den Namen zu übertragen, um vordefinierte RoIs
für den Nutzer intuitiv zu benennen.
Der Namensstring wird nicht in Anführungszeichen eingeschlossen.

@ LLOD gibt eine minimale Spezifikationsmenge der Mächtigkeit 1 für den
Ziel-LoD dieser RoI an. Größere Spezifikationsmngen sind aktuell nicht
implementiert. Dabei entspricht <?_res> der Differenz aus
#<number_of_levels> und Log2 des Nenners des Auflösungsfaktors (?_res>=0).
<precision> ist die maximale Genauigkeit in Bits, vom msb angefangen.
0<precision<=16 <color> hat den Wert 0 (Y-Kanal) oder 1 (Cb+Cr-Kanal).

@ PRIO gibt die Priorität für die akuelle RoI an.
<RoI_priority> ist Scheduling-Priorität für die RoI und gibt die relative
Datenmenge in bpp an (beliebiger Float).
Verschiedene RoIs dürfen die gleiche Priorität haben.
(AKTUELLE IMPLEMENTATION: <RoI_priority> ist integer und wird als
Bitplane-Differenz interpretiert)

@ APRIO priorisiert die Achsen des LoD-Vektorraumes für die Traversierung.
<?_resolution_prio>, <precision_prio> und <color_prio> sind als positive Integers
anzugeben.

AuthorCmd:
<step_cnt> gibt Schritt des RoI-Schedulers an, nach dem Cmd ausgeführt werden soll.
<keyword> ist Schlüsselwort des Kommandos. <params> sind die Parameter des
Kommandos.

Aktuell unterstützt (tbi = to be implemented):

```
STOPALL        (keine Parameter)
STOPROI        <roi_idx>
CONTROI        <roi_idx>
PRIOMOD        <roi_idx> <precPrio>
LLODMOD        <roi_idx> <lLoDResX> <lLoDResY> <lLoDPrec> <lLoDColr>
RESMOD         (tbi)
PRECMOD        (tbi)
COLRMOD        (tbi)
PRECINC        <roi_idx> <pos_increment>
XRESINC        <roi_idx> <pos_increment>
YRESINC        <roi_idx> <pos_increment>
PRECPRIOMOD    (tbi)
PRECPRIOINCDEC <roi_idx> <pos_or_neg_increment>

DEFRECT        <name> <EOL>
            <lLoDResX> <lLoDResY> <lLoDPrec> <lLoDColr> <precPrio> <EOL>
            <left> <top> <right> <bottom>

DEFELIPS    <name> <EOL>
            <lLoDResX> <lLoDResY> <lLoDPrec> <lLoDColr> <precPrio> <EOL>
            <left> <top> <right> <bottom>

DEFPOLY        <name> <EOL>
            <lLoDResX> <lLoDResY> <lLoDPrec> <lLoDColr> <precPrio> <n> <EOL>
            <X1> <Y1> <EOL>
```

```
            ...
            <Xn> <Yn>

DEFFISH        <N> <EOL>
            <XWire1> <YWire1> <EOL>
            ...
            <XWireN> <YWireN> <EOL>
            <Subsample0> <EOL>
            ...
            <Subsample_n/2+1> <EOL>
            <Prio0> <EOL>
            ...
            <Prio_n/2+1>
            Anmerkung:  Prio0, Subsample0: äußerster Ring
                     Prio_n/2+1, Subsample_n/2+1: Fokus

APPLICATION    <type_flag> <length> <0..255> {<0..255>}


BEISPIEL:
=========
Ein Hintergrundrechteck, eine Ellipse und ein
unregelmäßiges konkaves Viereck.
Farbauflösung in X- und Y-Richtung halbiert.
Farbverfeinerung schwächer priorisiert.
-------------------------------------------------
ROI19
c:\temp\image.pgm
4 1 8
3
@ NAME Background RoI
@ LLOD 4 4 16 0
@ PRIO 0
@ APRIO 4 4 1
@ RECT 0 0 511 511
@ NAME A Polygon
@ LLOD 4 4 16 0
@ PRIO 0
@ APRIO 4 4 1
@ POLY 4
322 35
488 40
414 155
388 85
@ NAME An ellipse
@ LLOD 4 4 16 0
@ PRIO 0
@ APRIO 4 4 1
@ ELLIPSE 184 232 187 223
1
0 PRECPRIOINCDEC 0 2
-------------------------------------------------
```