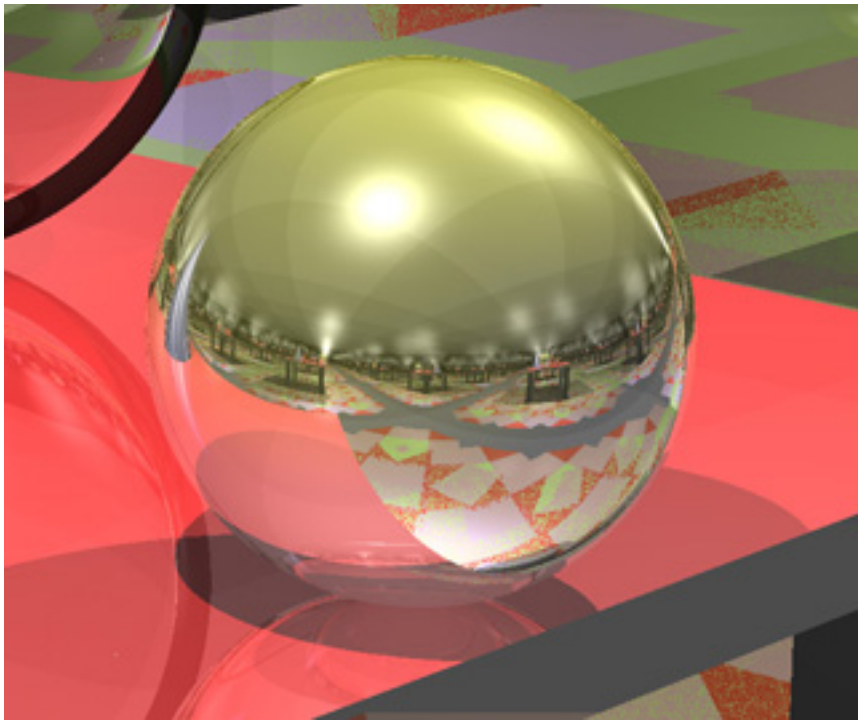
Algorithmen für beschleunigtes Rendering

Seminar - Susanne Krömker, Michael Winckler



Wintersemester 2006/2007

Ausarbeitungen, Stand 1. Mai 2007

Inhaltsverzeichnis

Vorwort

Das Seminar *Algorithmen für beschleunigtes Rendering* fand im Wintersemester 2006/2007 in der Arbeitsgruppe *Visualisierung und Numerische Geometrie* an der Universität Heidelberg statt. Verbindendes gemeinsames Interesse der Mitglieder der Gruppe, Studenten, Doktoranden, Hilfskräfte und Seminarleiter, war dabei der Spaß an Echtzeitverfahren, die ein direktes Feedback durch den Graphikrechner erlauben und gleichzeitig einen hohen Qualitätsstandard erfüllen.

Hier nicht ausgeführte Vorträge befassten sich mit dem *Marching Cube Algorithmus* (Michael Winckler), *Low Cost Laser Scanning und Matching Algorithmen* (Sven Molkenstruck, TU Braunschweig), sowie *Cg – A programming language for graphics cards*, (Somporn ChuaiAree). PDF-Dateien der Vortragsfolien sind auf Anfrage bei den Autoren zu bekommen.

Die hier zusammengestellten Seminarausarbeitungen sind in der Abfolge der Vorträge aufgeführt. Da die Themen inhaltlich nicht aufeinander aufbauen, ist jeder Teil auch separat verständlich.

Heidelberg, Mai 2007

Susanne Krömker[1]

[1]IWR, Universiät Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, kroemker@iwr.uni-heidelberg.de

Teil I

Displacement- und Bump Mapping

Andreas Diehl[2]

Abstract. *In dieser Ausarbeitung geht es um zwei Methoden, die mehr Details und somit mehr Realismus in die Computergrafik bringen. Namentlich sind dies Bump- und Displacement Mapping. Zuvor wird jedoch auf Heightmapping eingegangen, um nötiges Hintergrundwissen zu bereiten. Bump- und Displacement Mapping sind zwei relativ weit verbreitete und durchaus mächtige Techniken, die hier einführend erklärt und anhand von Beispielen verdeutlicht werden. Im Anschluss an die Vorstellung dieser beiden Themen werden diese Methoden miteinander verglichen, um Vor- und Nachteile im direkten Vergleich herauszuheben.*

1. Heightmapping

Mit *Heightmaps* bezeichnet man grundsätzlich ein äquidistantes Gitter aus Höheninformationen. In den meisten Fällen liegen Heightmaps als Bilder vor. Das Thema Heightmapping findet in der Computergrafik sehr viel Verwendung und in verschiedenen Einsatzgebieten jeweilige Unterschiede. Da Heightmapping aber wichtig für die zentralen Themen dieser Ausarbeitung ist, wird in diesem Kapitel darauf einführend eingegangen.

1.1. Typen von Heightmaps und Verwendung

Klassische Heightmaps sind in Graustufen gehalten. Die Graustufen kodieren somit die Höheninformation, wobei es Konvention ist, dass weiß, mit Farbwert 255, der höchste Punkt ist, und schwarz, mit Farbwert 0, der tiefste. Was hierbei hoch und tief heißt, wird in der jeweiligen Anwendung definiert. Abbildung 1 zeigt eine klassische Heightmap.

Manchmal findet man auch Hinweise auf farbige Heightmaps. In diesem Fall werden die RGB-Werte in einem Pixel als (x, y, z)-Vektor interpretiert und in den meisten Fäqllen als Oberfächennormale benutzt. Daher findet man diese farbigen Heightmaps vielmehr unter dem Begriff *Normalmap*. Für eine korrekte Anwendung müssen die RGB-Werte normiert sein. Abbildung 2 zeigt beispielhaft eine solche Normalmap. Hier erkennt man gut, dass die Farbe blau stark dominiert. Dies liegt daran, dass sie der z-Koordinate entspricht.

Die Verwendung von Heightmaps ist, wie eingehend schon angedeutet, sehr vielfältig, da sie als Datenspeicher für äquidistante Höheninformationen dienen. Ein Verwendungsgebiet, das man recht oft entdeckt, ist die Landschaftsmodellierung, aber vor allem auch Bump Mapping und Displacement Mapping.

[2]IWR, Universiät Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, krokodiehl@t-online.de

Abbildung 1. Heightmap mit einem Hügelzug (weiß) und einem kleinen Tal (schwarz)

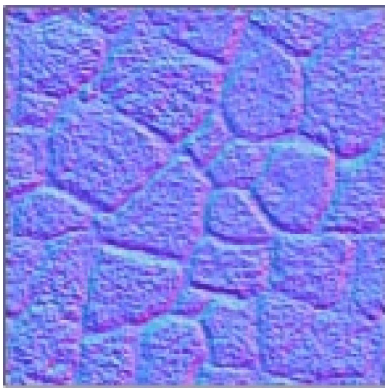


Abbildung 2. Beispiel einer Normalmap, die einen Steinboden zeigt

1.2. Erstellung von Heightmaps

Die Erstellung von Heightmaps kann sehr vielfältig sein und sich je nach Anwendungsgebiet unterscheiden. Hier möchte ich auf drei recht gängige Methoden eingehen.

1.2.1. Heightmap zeichnen

Eine recht intuitive Methode ist, dass man eigene Heightmaps mittels gängiger Bildverarbeitungssoftware selbst zeichnen kann. Dieser Vorgang kann natürlich beliebig komplex sein, jedoch erhält man auch mit wenig Aufwand deutliche Resultate. Die folgende Abbildung 3 zeigt eine selbst erstellte Heightmap, mittels *Paint Shop Pro 7.0*. Der Grauwert 125 wurde hier, als mittlerer Wert, für den Hintergrund gewählt. Später werden die Ergebnisse der Benutzung dieser Heightmap bildlich gezeigt.

Hier sieht man, dass jeweils Viertelkreise in den Ecken der Heightmap sind. Diese dienen dazu, dass man diese Map späterhin beliebig oft aneinander reihen kann. Dies ist jedoch für eine Heightmap nicht notwendig.

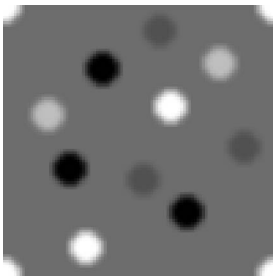


Abbildung 3. Selbst gezeichnete Heightmap

1.2.2. Heightmap rendern

Eine weitaus komplexere Methode, um zu einer Heightmap zu kommen, ist diese zu rendern. Anhand eines Tutorials, dass *3D Studio Max* benutzt, habe ich zunächst eine Ebene so modelliert, dass man eine einfache Landschaft erkennen kann, wie die Abbildung 4 zeigt.

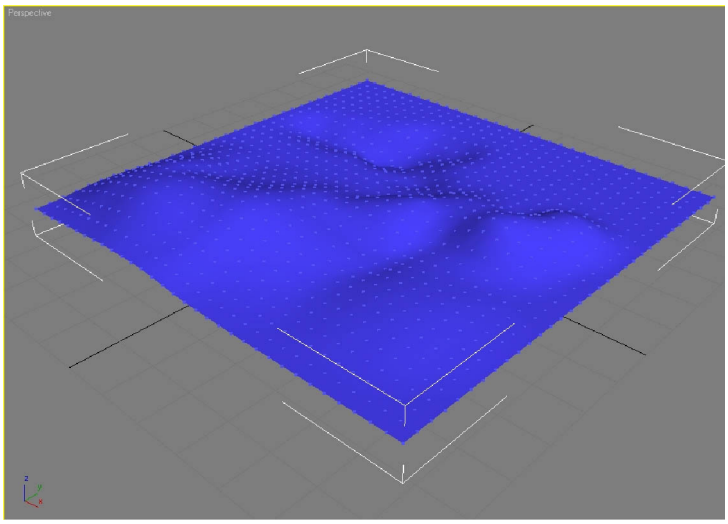


Abbildung 4. Modellierte Landschaft aus einer Ebene (3D Studio Max 6.0)

In weiteren Schritten, die hier nicht im Detail erklärt werden, wird dieses Modell nun mit einem Farbverlauf versehen, der von schwarz bis weiß geht. Hierbei muss man darauf achten, dass dieser Farbverlauf tatsächlich so an das Modell angepasst wird, dass der tiefste Punkt tatsächlich schwarz, und der höchste Punkt weiß ist. Rendert man anschließend dieses Modell in der Aufsicht, so erhält man eine gültige Heightmap. Abbildung 4 zeigt im übrigen das Beispiel aus Abbildung 1. Die genauen Schritte sind anhand eines Tutorials in [1] erklärt.

1.2.3. Heightmap aus Texturen

Da Heightmaps oftmals zusammen mit normalen Texturen benutzt werden, ergibt sich die Notwendigkeit, beide aufeinander abzustimmen. Daher gibt es Wege, aus einer vorhandenen Textur eine Heightmap zu generieren. Dies kann automatisch oder mittels eines Bildverarbeitungsprogramms gemacht werden.

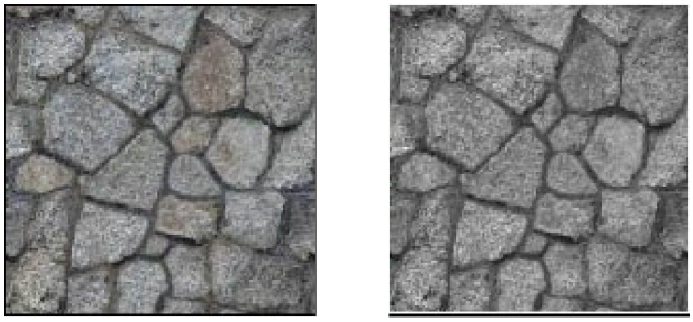


Abbildung 5. Textur (links) und daraus erstellte Heightmap (rechts)

Abbildung 5 zeigt ein Beispiel, das mit *Photoshop* erstellt wurde, nachzulesen in [2]. Die Normalmap in Abbildung 2 ist übrigens ebenfalls aus dieser Textur erstellt worden.

2. Bump Mapping

Bump Mapping ist eine vergleichsweise alte Methode, die erstmals 1978 von James Blinn vorgestellt wurde. Sie ist aufgrund der recht einfachen Anwendung sehr verbreitet und gehört mittlerweile zu den Basismethoden im Bereich Mapping und Texturing [3]. In diesem Kapitel soll die grundlegende Idee von Blinn erklärt und schließlich anhand von Beispielen der deutliche Effekt dieser Technik gezeigt werden. Der Begriff Bump Mapping kommt vom englischen Wort für Beule (bump).

2.1. Idee der Normalenänderung

Blinns Methode benutzt die Tatsache, dass die Wahrnehmung von Oberflächenstruktur allein auf Lichtreflektion und brechung zurückzuführen ist. Genau genommen ist die Oberflächennormale ausschlaggebend für die Lichtreflektion, wie in Abbildung 6 verdeutlichen soll:

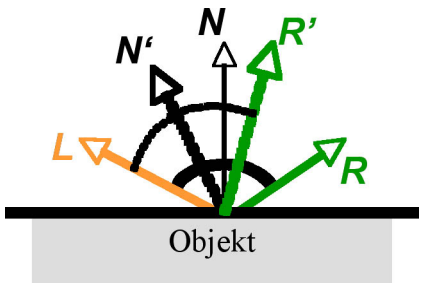


Abbildung 6. Normale und ihre Auswirkung auf die Lichtreflektion

Man erkennt, dass der Winkel zwischen dem Vektor L zur Lichtquelle und der Oberflächen Normalen N dem Reflektionswinkel gleicht, der das reflektierte Licht R bestimmt. Verändert man nun also die Normale zu N' (gestrichelte Darstellung), so verändert sich auch die Lichtreflektion zu R'. Bedeutend hierbei ist aber, dass sich weder die Geometrie, sprich Lage und Form des Objektes noch der Lichtquelle verändert. Blinns Methode verändert also allein die Normale.

Bevor man nun jedoch eine Normale direkt ändern kann, muss man die betreffende Fläche parametrisieren. Diese Parameter (u, v) geben nun einen Punkt auf der Oberfläche an, mittels einer

Parametrisierungs-Funktion P. Die analytische Normale in einem Punkt (u, v) lässt sich somit über das Kreuzprodukt errechnen:

$$N = P_u \times P_v$$

wobei P_u und P_v jeweils partielle Ableitungen der Funktion $P(u, v)$ sind. Es ergibt sich schließlich eine einfache Formel für das Verändern einer Normalen im Punkt (u, v):

$$N' = N + \frac{B_u(N \times P_v) - B_v(N \times P_u)}{\|N\|}$$

Die neue Normale N' ergibt sich aus der eigentlichen Normalen mit einem neuen, normierten Vektor, der die Höheninformationen an der Stelle (u, v) benutzt. Die Funktion $B(u, v)$ steht hier für Bump Map und erhält eine Höheninformation für den Punkt (u, v). In diesem Fall kann eine Bump Map sowohl eine Heightmap, eine Normalmap oder auch eine mathematische Funktion, so genannte Noise-Funktion [4], sein. Grafisch verdeutlicht wird die Neuberechnung der Normalen in Abbildung 7:

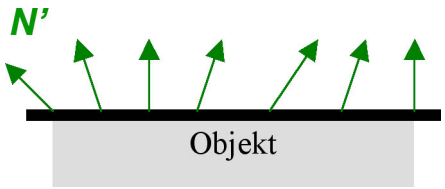


Abbildung 7. Veränderte Normalen beim Bump Mapping

Man erkennt, dass die neuen Normalen N' nun dafür sorgen, dass die Lichtberechnung auf dieser Oberfläche anders abläuft. Für einen Betrachter entsteht so der Eindruck von Oberflächenstruktur, ohne dass diese tatsächlich vorhanden ist. In dieser Simulation von Unebenheiten steckt der große Vor- aber auch Nachteil von Bump Mapping, wie man in Kapitel 4. sehen wird.

2.2. Beispiele

Das folgende Beispiel zeigt die Anwendung der selbst gezeichneten Heightmap aus Abbildung 3 als Bump Map auf einer Kugel. Deutlich sichtbar sind die einzelnen Erhebungen und Vertiefungen, die allein durch die Lichtberechnung entstanden sind. Das Grundobjekt ist nach wie vor eine Kugel:

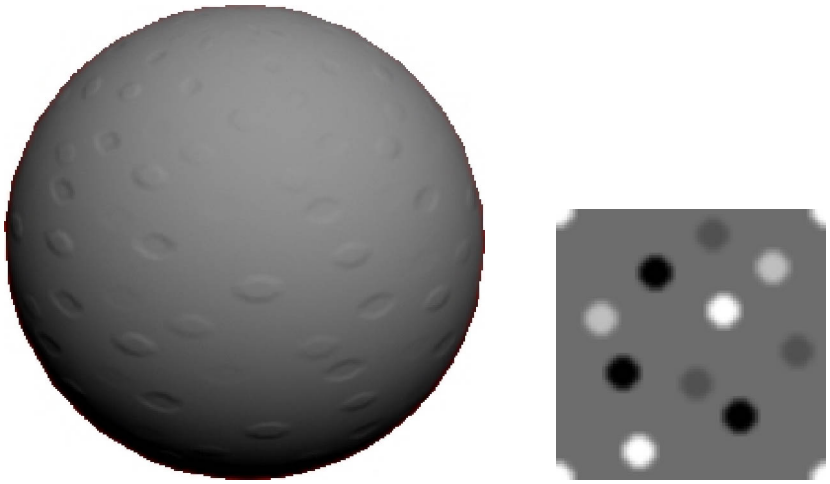


Abbildung 8. Bump Map aus Abbildung 3 (rechts) auf einer Kugel

Ein anderes Beispiel soll den enormen Effekt von Bump Mapping zeigen. Hier wurde versucht, eine Orange zu rendern:

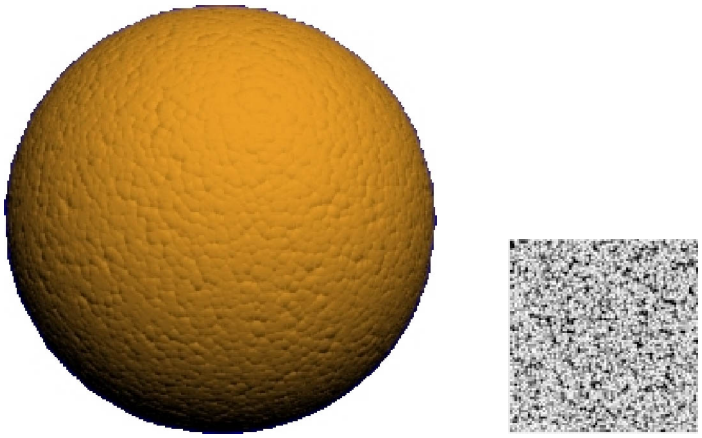


Abbildung 9. Orange durch Bump Mapping (Heightmap rechts)

Man erkennt die unebene Struktur der Orangenhaut. An diesem Beispiel kann man sich gut verdeutlichen, welchen Aufwand es bedeuten würde, die einzelnen Unebenheiten über die Geometrie direkt zu modellieren. Bump Mapping sorgt hier für einen deutlich realistischeren Effekt ohne einen wirklichen Mehraufwand bei Modellierung und Rechenzeit.

3. Displacement Mapping

Displacement Mapping wurde 1984 von Cook erstmals vorgestellt, doch erst drei Jahre später, 1987, implementiert. Der Grund liegt darin, dass diese Methode, die vor allem die im Folgnden aufgefuhrten Nachteile von Bump Mapping beheben sollte, zwar von der Idee her einfach, aber in der Umsetzung recht komplex ist.

Die grundsätzliche Idee hinter Displacement Mapping ist, dass man direkt die Oberfläche, sprich die Geometrie des Objektes verändert. Dies geschieht anhand von Höheninformationen aus z.B. einer

Heightmap, die hier im speziellen Fall auch Displacement Map genannt wird. Die einfache Formel lautet:

$$\text{neuer Punkt} = \text{alter Punkt} + \text{Höhe} \cdot \text{Normale}$$

Mathematisch ausgedrückt, lautet die Formel

$$P'(u,v) = P(u,v) + D(u,v)\frac{N(u,v)}{\|N(u,v)\|},$$

wobei der Punkt (u, v) durch eine Parametrisierung der Fläche bestimmt wird, und $D(u, v)$ die entsprechende Höheninformation aus einer Displacement Map ist.

3.1. Erste Implementierung

In der ersten Implementierung 1987 wurden so genannte Mikropolygone benutzt. Ein großes Problem beim Displacement Mapping ist nämlich, dass Flächen weiter unterteilt werden müssen, damit sie entsprechend einer Heightmap verschoben (displaced) werden können. Der erste Mikropolygon-Ansatz war nur für Scanline-Rendering benutzbar und unterteilte eine Fläche in weitere Unterflächen, die so genannten Mikropolygone, bis diese schließlich kleiner als ein Bildpixel waren.

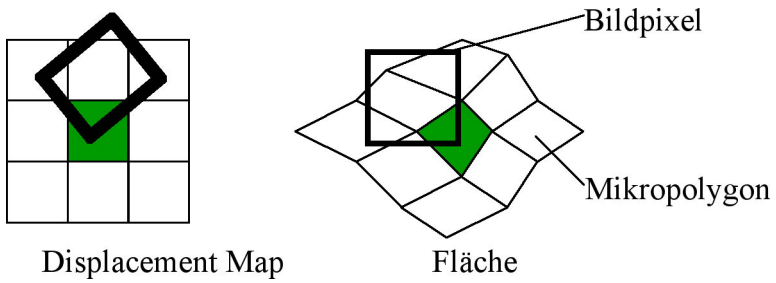


Abbildung 10. Mikropolygon-Ansatz

Für Raytracing war dieser Ansatz aufgrund der ernormen Geometriezunahme völlig unpraktikabel. Ab etwa 1995 gab es erste Ansätze für den Einsatz von Displacement Mapping in globalen Beleuchtungsmodellen. Eine Idee ist, die Strahlen des Raytracers nicht mit der Geometrie, sondern der Displacement Map direkt zu schneiden. Ein Nachteil hierbei ist jedoch, dass diese Schnittberechnung sehr komplex ist. Ein anderer Ansatz nutzt die räumlichen Zusammenhänge der Geometrie aus, indem sie die benachbarten Polygone cached. Diese Methode ist zwar sehr aufwendig zu implementieren, läuft jedoch relativ performant [5] [6].

3.2. Weitere Ansätze

Es gibt eine Reihe von weiteren Ansätzen, die über die Zeit hinweg entstanden sind, um Displacement Mapping effizienter zu machen. Auf diese soll hier jedoch nur grob eingegangen werden:

- *Tesselation* wurde etwa 1999 für Displacement Mapping benutzt. Hier werden ebenfalls Mikropolygone erzeugt, die jedoch in einem Präprozess generiert werden.

- *Displaced Subdivision Surface* entstand ca. 2000 und nutzt ein einfaches Kontrollgitter (Mesh), welches displaced wird. Dieser Ansatz wird vor allem in der 3D-Modellierung für Animationen verwendet.
- *Adaptive View Dependend Tesselation* wurde etwa 2000 von Doggett und Hirche entwickelt. Hier werden Flächen rekursiv unterteilt, je nachdem wie dicht Höheninformationen vorliegen. Es ist ein relativ dynamisches Verfahren, welches aber vor allem Probleme mit der Rekursionstiefe hat.

Siehe die Literaturstellen [5] [6].

3.3. Beispiele

Es werden nun ein paar Beispiele von Displacement Mapping gezeigt. Zunächst wird wieder die selbst erstellte Heightmap aus Abbildung 3 benutzt. Der Effekt von Displacement Mapping ist hier deutlich sichtbar. Es muss jedoch klar sein, dass die Interpretation einer Heightmap relativ ist, sprich wie sich höchster bzw. tiefster Punkt auf die Geometrie auswirkt, wird vom Anwender bestimmt.

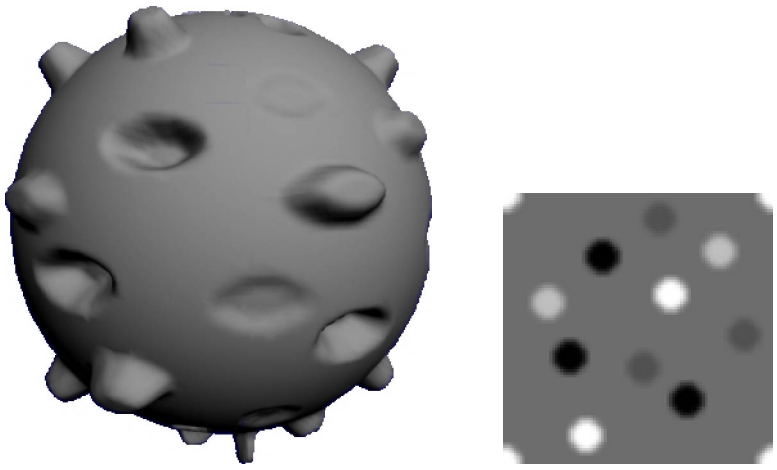


Abbildung 11. Bekanntes Beispiel mit Displacement Map

Das nächste Beispiel zeigt die Heightmap aus Abbildung 1, angewandt auf eine einfache Ebene. Dies entspricht der Arbeitsweise der Landschaftsmodellierung: man erkennt, wie aus der flachen Ebene eine hügelige Landschaft allein durch die Anwendung einer Heightmap wird.

Der Effekt von Displacement Mapping ist, im Vergleich zum Bump Mapping, sehr deutlich sichtbar. Im folgenden Kapitel werden die beiden Methoden nun direkt verglichen, um die jeweiligen Vor- und Nachteile hervorzuheben.

4. Vergleich

Wie man aus den Kapiteln 2. und 3. anhand der Beispiele schon erkennen konnte, haben Bump und Displacement Mapping ähnliche Resultate, mit jedoch großen Unterschieden. Displacement Mapping, historisch gesehen, entstand nach Bump Mapping und sollte vor allem dessen Nachteil überwinden. Dieser Nachteil soll in Abbildung 13 verdeutlicht werden:

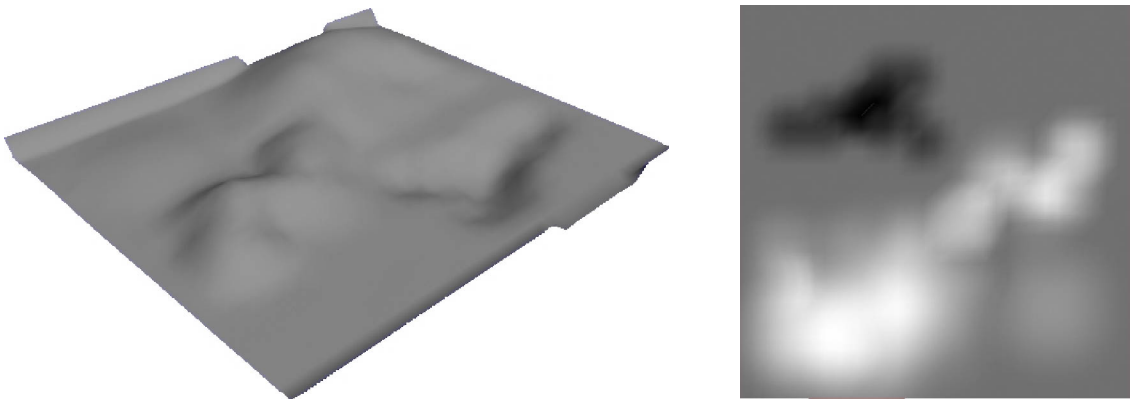


Abbildung 12. Displacement Mapping bei der Landschaftsmodellierung

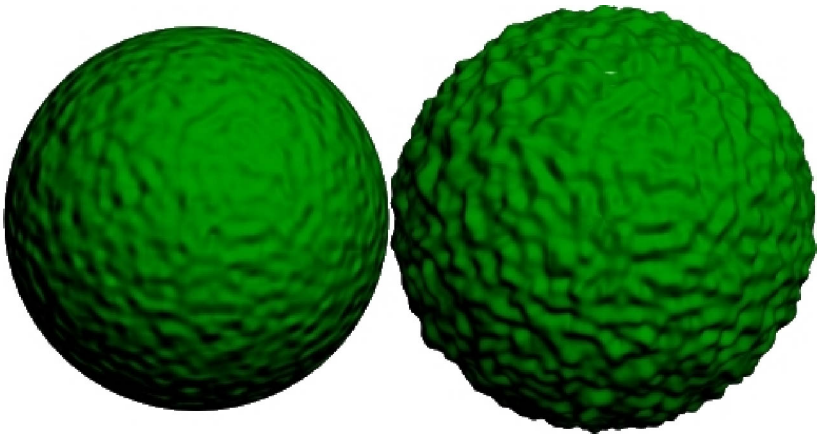


Abbildung 13. Bump und Displacement Mapping im direkten Vergleich

Beide Objekte basieren auf einer einfachen Kugel und benutzen die gleiche Heightmap. Links zeigt sich der Effekt von Bump Mapping, rechts von Displacement Mapping. Abgesehen von der Tatsache, dass die rechte Kugel größer ist, da ihre tatsächliche Geometrie durch das Mapping nach außen verschoben wurde, sieht man an der Silhouette der Objekte den größten Nachteil von Bump Mapping. Da hier die Oberflächenstruktur nur simuliert wird, erkennt man am Rand das zugrunde liegende Objekt. Es soll an dieser Stelle jedoch erwähnt werden, dass das Rendern der linken Kugel in Sekunden ging, wohingegen das Berechnen der rechten Kugel mit der Displacement Map fast eine Minute gedauert hat.

Die Vor- und Nachteile aufgezählt, erhalten wir folgende Zusammenfassung. Bump Mapping wird hier mit BM abgekürzt, Displacement Mapping mit DM:

- *Detailgrad:* BM simuliert Details nur, DM indes modelliert Details und ist damit realistischer, zumal die veränderte Geometrie auch einen anderen Einfluss auf die Beleuchtung nahe liegender Objekte hat.
- *Einfachheit:* BM beeinflusst lediglich die Lichtberechnung und ist daher eine sehr einfache Methode, die ebenfalls sehr gute Resultate liefert. Die Ergebnisse von DM sind tatsächlich

weitaus besser, jedoch zu einem sehr hohen Preis. Methoden, die DM praktikabler machen, sind meist komplex und schwierig umzusetzen.

- *Performanz:* BM ist in Echtzeit machbar und auf neuerer Graphikhardware bereits integriert. Für DM gilt dies (noch) nicht.
- *Datengrundlage:* Beide Techniken nutzen die gleiche Grundlage für ihre Höheninformationen. Seien dies Heightmaps oder Normalmaps.
- *Fehlertoleranz:* BM arbeitet fehlerfrei. DM indes nicht. Hier dürfen die Höheninformationen nicht zu stark interpretiert werden, da sonst sehr leicht Löcher in der Geometrie entstehen können oder versetzte Objekte andere Objekte plötzlich durchdringen könnten.

Literatur

[1] `http://www.ogre3d.org/wiki/index.php/3dsmax_Heightmaps` (Stand 08.11.2006)

[2] `http://www.mapping-tutorials.de/forum/showthread.php?t=167&goto=nextoldest` (Stand 08.11.2006)

[3] D. S. Ebert et al., *Texturing & Modeling, A Procedural Approach*, 3. Edition, 2003, Morgan Kaufmann Publishers, San Francisco

[4] F. S. Hill, Jr., *Computer Graphics, Using Open GL*, 2. Edition, 2001, Prentice Hall Inc, New Jersey

[5] Kimmu Karhu, *Displacement Mapping*, Helsinki University of Technology, 2002 (`http://www.tml.tkk.fi/Opinnot/Tik-111.500/2002/paperit/kimmo_karhu.pdf`)

[6] C. Chen, L. Huang, *Uniform Displacement Mapping*, `http://www.cs.berkeley.edu/~hling/courses/cs284/final.html` (Stand 09.11.2006)

Teil II

Binary Space Partitioning

Ronald Lautenschläger[3]

Abstract. *Das Binary Space Partitioning ist populär aus Ego-Shootern wie DOOM oder Quake von id-Soft™. Der Aufwand des Renderns einer Szene steigt mit der Anzahl der beteiligten Objekte. Es kommt jedoch vor, dass manche Objekte andere verdecken, so dass diese für den Betrachter nicht sichtbar sind und deswegen auch nicht berücksichtigt werden müssen. Beim Binary Space Partitioning kommen binäre Bäume (BSP-TREES) zum Einsatz, die man als beschleunigende Datenstruktur einsetzt, um beim Zeichnen einer Szene nur die sichtbaren Objekte berechnen zu müssen. Die Beschleunigung wird durch eine hierarchische Untergliederung des Objektraums erreicht. Hierdurch weiß man, welche Objekte von der aktuellen Kameraposition sichtbar sind und welche nicht.*

1. BSP-Bäume

1.1. BSP-Baum Definition

Ein BSP-Baum ist eine Datenstruktur (Binärbaum), die sich durch eine hierarchisch-rekursive Unterteilung auszeichnet. Ein n-dimensionaler Raum wird in Halbräume unterteilt. Durch gute Sortier- und Klassifikationseigenschaften von Binärbäumen sind BSP-Bäume vielseitig einsetzbar.

1.2. Konstruktion eines BSP-Baums

Zur Veranschaulichung soll anhand eines 2D-Beispiels ein BSP konstruiert werden. Man unterteilt einen 2D Raum mit Geraden in Halbräume. Im einfachsten Fall ist dieser Raum durch vier Wände definiert. Gegeben sei ein Viereck, das nach und nach durch Linien (P1 und P2) unterteilt wird.

Abbildung 14 zeigt die schrittweise Aufteilung des Raumes: Im Schritt 1 haben wir noch keine Unterteilung. Im Schritt 2 hingegen sind durch die Teilung zwei neue Bereiche (Halbräume) entstanden (B, C). Hier wird zwischen Vorder- und Rückseite unterschieden. Im Schritt 3 wird einer dieser Halbräume wiederum in (D, E) unterteilt.

Die Baumstruktur verdeutlicht, wie die hierarchische Einordnung der neu entstandenen Objekte vorgenommen wird. Im folgenden Beispiel soll verdeutlicht werden, wie BSP dazu dient, die korrekte Zeichenreihenfolge der Objekte zu bestimmen. Wie schon im ersten Beispiel unterteilt eine Linie P1 unser Objekt in zwei Halbräume. Die einzelnen Teilstücke des Objekts (A,..,H) werden je nach Lage zu Teilungslinie in Vorder- und Rückseite eingeordnet, wie im Baum zu sehen ist.

[3]IWR, Universiät Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Ronald.Lautenschlaeger@web.de

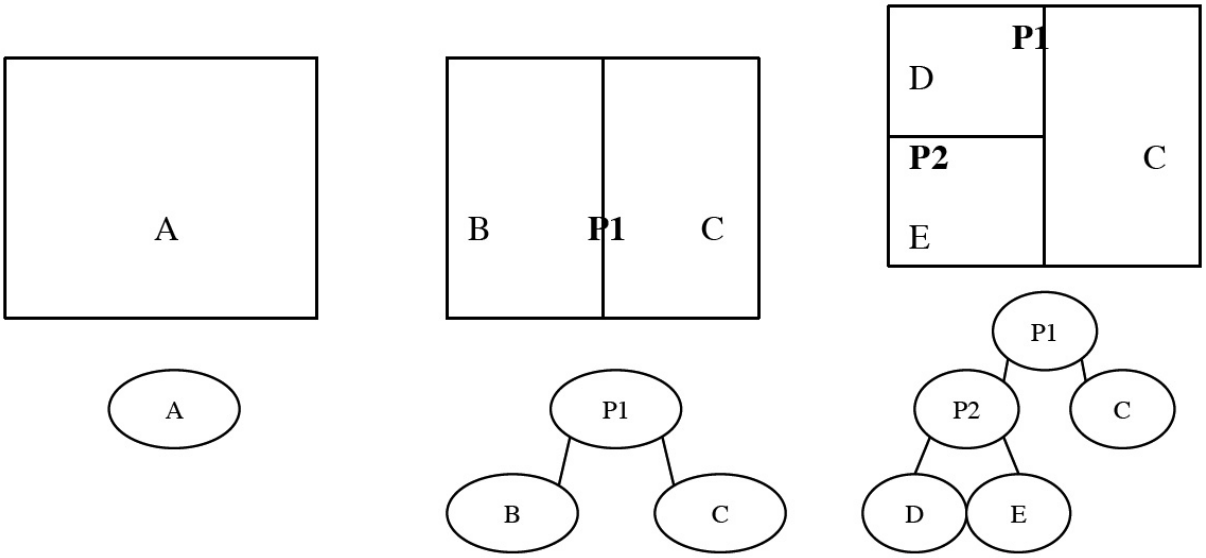


Abbildung 14. Beispiel für Unterteilung in 2D mit dazugehörigen Bäumen

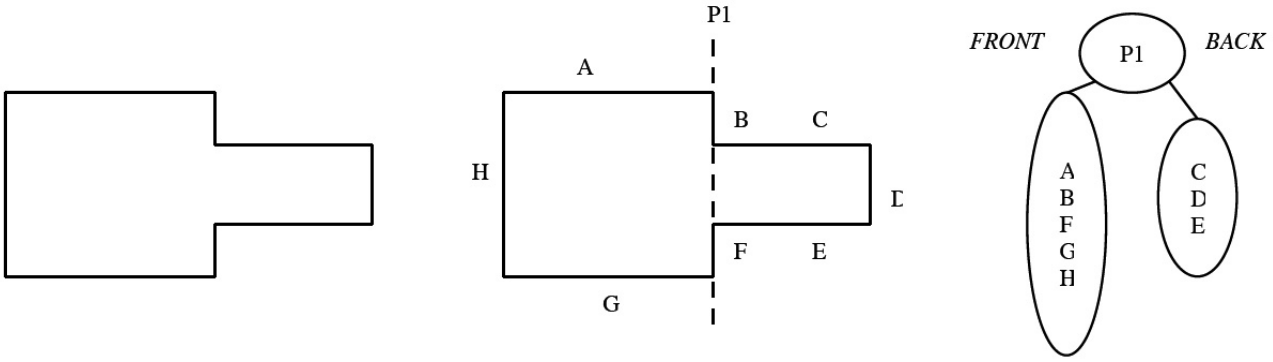


Abbildung 15. Unterteilter Raum in 2D

Man stelle sich die Abbildung 16 als Grundriss einer Szene vor. Der Stern sei ein Betrachter mit einem Sichtbereich, der durch die graue Fläche repräsentiert wird. An diesem Beispiel wird schnell klar, wie die Reihenfolge der zu zeichnenden Objekte aussehen muss. Wenn sich der Betrachter auf der Vorderseite befindet, ist es nötig, zuerst die Rückseite zu zeichnen und umgekehrt.

1.3. BSP-Algorithmus

Das Vorgehen ist wie folgt:

BSP_BAUM (Set_of_Polygons)
1) Wähle eine Teilungsebene
2) Teile das Polygon-Set mit der Ebene
3) Verfahre analog rekursiv mit den neu entstandenen Sets

Die Wahl der Teilungsebene ist abhängig von Einsatzbereich bzw. den Kriterien der Effizienz, die für

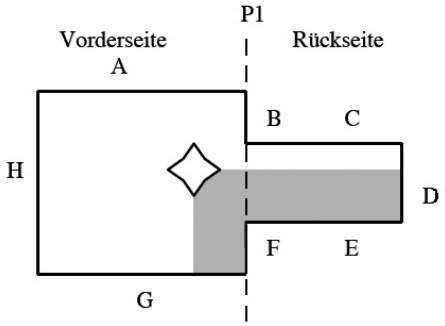


Abbildung 16. Gleicher Raum wie in Abb. 15 mit Betrachter und Sichtbereich

den jeweiligen Einsatzbereich veranschlagt werden. Ein Beispiel hierfür wäre, dass man als Ziel einen möglichst balancierten d.h. ausgewogenen Baum erhält, was beim Raytracing von hoher Bedeutung sein kann. Durch die Wahl der Teilungsebene kann man aber auch die Anzahl der resultierenden Polygone beeinflussen, denn durch Schneidung einer Teilungsebene mit einem Polygon entstehen mindestens zwei neue Polygone. Jedes Polygon wird im Bezug auf die Teilungsebene einem Ast des Baumes zugeteilt. Schneidet ein Polygon die Ebene, wird es aufgespalten und die Teilstücke der jeweiligen Seite zugeordnet. Die Teilungsebenen sind in der Regel entlang den Achsen ausgerichtet *(Axis-Aligned-BSP)* oder orthogonal.

Die Terminierung der Baum-Konstruktion ist abhängig vom Anwendungsfall bzw. der Art des BSP. Gründe zur Terminierung können sein, dass Polygone in einem Knoten unter einem gewissem Maximum liegen, dass eine vorgegebene maximale Baum-Tiefe erreicht wurde oder jedes Polygon einem Knoten zugeordnet wurde.

1.4. Teilung eines Polygons durch eine Ebene

Die Teilung von konvexen Polygonen ist ein einfacher Vorgang, weil immer zwei neue entstehen. Im Gegensatz dazu ist die Teilung von sich selbst überdeckenden konkaven Polygonen ein großes Problem im BSP.

Für die Teilung eines Polygons muss dessen Lage genau bestimmt werden. Um zu testen, ob alle Punkte eines Polygons auf einer Seite der Teilungsebene liegen (Punkte der Ränder des Polygons) benutzt man den *Front-Back-Test*. Dieser ermittelt die Distanz von der Teilungsebene zum Punkt entlang der Normalen der Ebene. Durch ein positives oder negatives Ergebnis des *Front-Back-Tests* weiß man, ob der aktuelle Punkt vor oder hinter der Teilungsebene liegt.

1.5. Unterteilungsbeispiel für 2D-Raum

Im Folgenden wird an einem Beispiel die schrittweise Unterteilung und der zugehörige Baum dargestellt. Muss ein Knoten unterteilt werden, taucht er danach mit dem Zusatz f (front) oder b (back) auf.

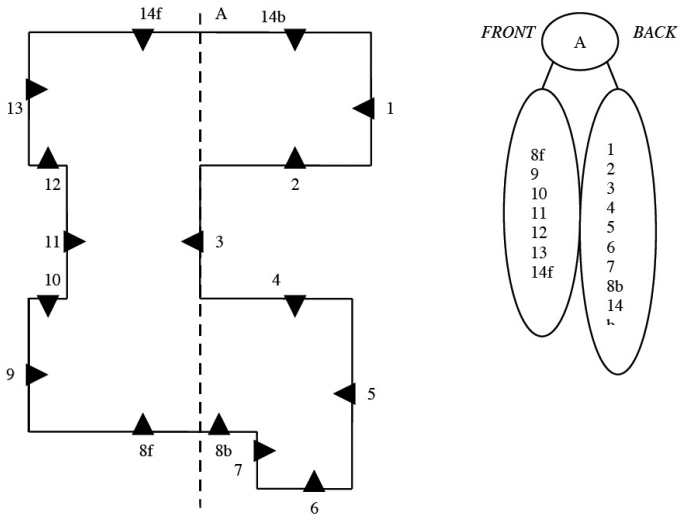


Abbildung 17. Unterteilung Schritt #1

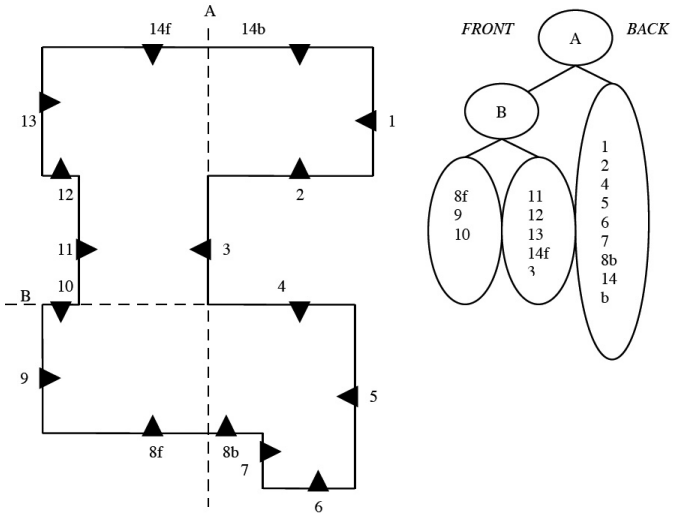


Abbildung 18. Unterteilung Schritt #2

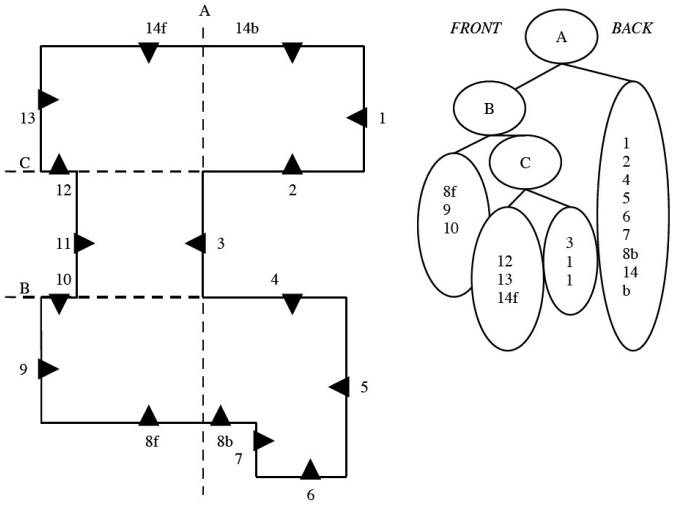


Abbildung 19. Unterteilung Schritt #3

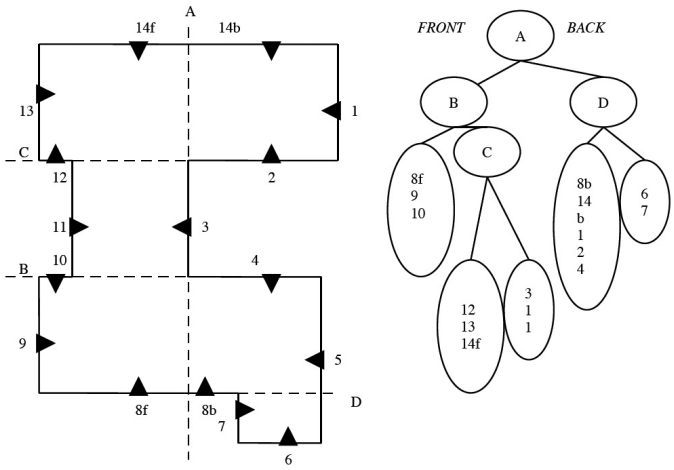


Abbildung 20. Unterteilung Schritt #4

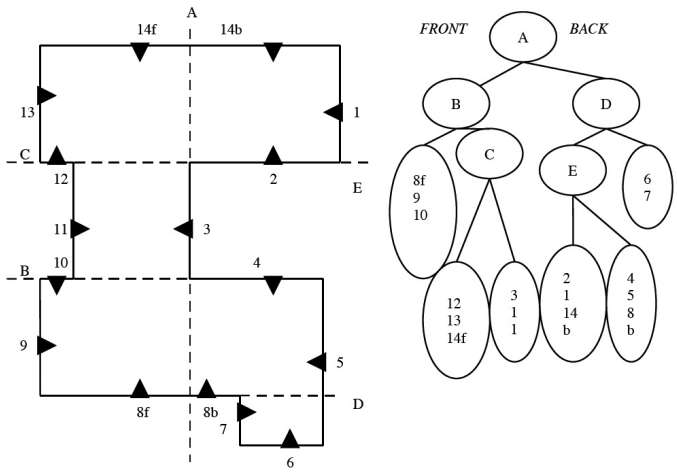


Abbildung 21. Unterteilung Schritt #5

1.6. Anwendung im 3D-Raum

Im 3D-Raum funktioniert die Partitionierung genau wie in 2D mit dem Unterschied, dass zur Unterteilung des Raumes eine Ebene verwendet wird. Der Raum setzt sich aus einzelnen Polygonen zusammen.

1.7. BSP-Implementierung

```
struct BSP_node{

	int nodeID; // Eindeutiger Identifier
	Plane3d partition; // Teilungsebene
	BSP_node *frontnode; // Zeiger auf Front der Ebene
	BSP_node *backnode; // Zeiger auf die Rückseite der Ebene
	bool leaf; // Blatt ja, nein
	int numpolys; // Anzahl der Polygone in diesem Knoten. = 0, wenn Blatt
	Polygon3d *nodepolylist // Zeiger die Liste mit Polygonen des Knoten
}
```

Dieses Beispiel zeigt eine C-Struktur, die ein Knoten des BSP-Baumes repräsentieren soll. Die zugehörigen Attribute sind eine eindeutige Kennzeichnung, eine Teilungsebene, Zeiger auf die Vorder- und Rückseite der aktuellen Ebene (Kindknoten), ein boolescher Wert, der auf *wahr* gesetzt wird, wenn es sich um ein Blatt handelt, die Anzahl der Polygone in dem aktuellen Knoten und einen Zeiger auf eine Liste, die alle Polygone des Knotens beinhaltet. Der rekursive Aufbau des Baumes funktioniert wie folgt:

```
BuildBspTree (BSP_node *node){

Neue Teilungsebene wählen aus der Polygonlist des Knotens            (node-
>nodepolylist)
Wenn nicht weiter unterteilbar, STOPP und markiere Knoten als Blatt
             (node->leaf = true)
Zähle Polygone vor bzw. auf der Teilungsebene
Speicher für Polygone allokieren
Vorder- und Hinterknoten node->frontnode node->backnode bestimmen
Teile die Polygone in „Vorder" und "Rückseite„ ein
Falls Polygone eindeutig, Vorder- bzw. Hinterknoten zuweisen
Falls Polygon Ebene schneidet, unterteilen und jedes Stück entsprechendem Knoten
zuweisen
Falls ein Polygon in der Ebene liegt, Normalenvergleich Polygon/Ebene. Wenn Normalen
gleich sind Polygon dem Frontknoten zuweisen
BuildBspTree(node->frontnode);
BuildBspTree(node->backnode);
}
```

2. Hidden-Surface-Removal

Das *Hidden-Surface-Removal* (HSR) ist ein beliebtes Einsatzgebiet von BSP-Bäumen im 3D-Raum. Hierbei geht es um die Entfernung verdeckter Flächen, die für den Betrachter nicht sichtbar sind. BSP-Bäume bieten einen effizienten Weg der Polygonvorsortierung mittels *Depth-First–Baumdurchlauf*. Dieser Vorteil lässt sich mit dem *Painters Algorithm Front-To-Back–Scanline* sehr gut ausnutzen.

3. Painters Algorithm

Der *Painters Algorithm* geht folgendermaßen vor:

1. *Beginne bei Wurzel und bestimme Blickpunkt im Bezug auf die Teilungsebene*
2. *Zeichne Unterbaum des vom Blickpunkt am weitesten entfernten Knoten*
3. *Zeichne den näher liegenden Unterbaum*
4. *Wiederhole rekursiv für jeden Unterbaum*

Die Idee hinter diesem Algorithmus ist, dass er Polygone, die weit vom Betrachter entfernt sind, zuerst zeichnet und danach die näher liegenden. Verdeckte Flächen werden überschrieben, sobald die näher liegenden Flächen gezeichnet werden. Einzige Bedingung ist, dass eine Fläche existieren muss, die die Polygone eindeutig voneinander trennt, sonst ist keine korrekte Darstellung mit *Painters Algorithm* möglich. BSP-Bäume sind ideal für diesen Algorithmus, weil die Teilung schwieriger Polygone schon Bestandteil der BSP-Konstruktion ist. Um den Inhalt des Baumes zu zeichnen, wird ein *Back-To-Front*–Baumdurchlauf vollzogen.

4. Scanline

Der *Scanline-Algorithmus* durchläuft den Baum in *Front-To-Back*–Richtung. Dabei wird jedes Primitiv zeilenweise auf überdeckung überprüft. Eine so genannte *Write-Mask* verhindert mehrmaliges Zeichnen der Pixel. In dieser Maske sind so genannte *Pixel-SPANs* enthalten, wovon es *n-SPANs* für

jede Scanline geben kann. Die Idee dahinter ist, dass nur diese *SPANs* geschrieben werden. Für diesen Algorithmus ist eine effiziente Methode der Maskierung nötig. Der Aufwand für *Front-To-Back* und *Back-To-Front*–Baumdurchlauf ist gleich. Die Scanline eignet sich optimal für die Berechnung komplexer Lichtmodelle, weil sie im Gegensatz zum *Painters Algorithm*, der jedes Pixel mehrmals auswertet, dies nur einmal machen muss.

5. Dynamische Szenen

Normalerweise ist das BSP nur für statische Objekte einer Szene gedacht. Da *Hidden-Surface-Removal* relativ einfach funktioniert, wäre dies auch für dynamische Inhalte einer Szene denkbar.

Eine Möglichkeit wäre, einen BSP-Baum zu erstellen, der die statischen Objekten einer Szene enthält, und dies für jeden Frame zu machen. Danach fügt man die dynamischen Inhalte ein. Das Problem hierbei ist, dass diese einfache Implementierung schon einen beachtlichen Rechenaufwand verursachen kann. Wenn ein dynamisches Objekt von jedem statischen durch eine Ebene getrennt ist, kann dieses als einzelner Punkt aufgefasst werden und dies unabhängig von der Komplexität. Für jedes dynamische Objekt kommt nur ein neuer Knoten hinzu. Man beachte: Bei statischen Objekten ist es ein Knoten für jedes Polygon im Objekt. Beim Baumdurchlauf muss jeder dynamische Knoten wieder in das originale Objekt expandiert werden. Der Vorteil, den BSP als Beschleunigungsstruktur mit sich bringt, entfällt hier gänzlich.

Das Einfügen eines dynamischen Objekts als Punkt ist eine einfache Operation. Ein dynamischer Punkt ist in der BSP-Baum Hierarchie als Kind von einem statischen oder dynamischen Knoten möglich. Wenn der Vater statisch ist, wird ein Front/Back-Test vorgenommen und der Knoten einsortiert. Ist der Vater dynamisch, so werden Distanzen zum Blickpunkt verglichen (Punkt teilt nicht 3D-Raum). Da Punkte nicht geteilt werden können, müssen diese deshalb eindeutig von anderen Objekten unterscheidbar sein, was eine Abgrenzung durch eine Teilungsebene notwendig macht. Ein Alternatives Verfahren wäre beim Einfügen eines dynamischen Knotens eine Ebene zu erzeugen, deren Normale der Vektor vom Blickpunkt zum Punkt ist. Das Erzeugen einer solchen Hilfsebene ist nur mit einem geringen Aufwand verbunden.

Das Leeren des Speichers am Ende eines Frames ist durch die Baumstruktur mit dynamischen Knoten ein verhältnismäßig einfacher Vorgang. Da ein statischer Knoten nicht Kind eines dynamischen Knoten sein kann, weil diese immer erst nachträglich hinzugefügt werden, können alle Unterbäume eines dynamischen Knoten gleichzeitig mit diesem entfernt werden.

6. Beschleunigtes Raytracing

Raytracing ist ein räumliches Einordnungsproblem. Ähnlich zu *Hidden-Surface-Removal* mit BSP-Baum wird ein *Front-To-Back*–Baumdurchlauf vollzogen. Ein großer Unterschied zu anderen BSP-Anwendungen ist, dass die Spaltung der Polygone für das Raytracing nicht nötig ist. Das Kriterium für die Ebenenwahl ist nur die Balance des Baumes (Ausgewogenheit der Teilbäume). Die Balance des BSP-Baumes ist deshalb entscheidend für die Performanz, weil beispielsweise beim rekursiven Raytracing der Aufwand zählt, der benötigt wird, um von der Wurzel zu einem beliebigen Blatt des Baumes zu gelangen.

7. Kollisionserkennung mit BSP

Die Bewegung eines Objekts wird in diskreten Zeitintervallen wie folgt berechnet.

$$P = P_0 + (t \cdot v).$$

Es wird zum Beispiel ein Berechnungsschritt pro Sekunde angenommen. Man berechnet die Endpunkte der Bewegung $(P_1,\ P_2)$.

$$P_1 = P_0 + v,\ \ P_2 = P_0 + (2 \cdot v)$$

Danach erfolgt das Einordnen der Endpunkte $P_1,\ P_2$ in Bezug zum BSP-Baum.

Fall 1: Wenn der Punkt P_1 außerhalb, P_2 innerhalb eines Objekts liegt, so ist eine Kollision aufgetreten.

Fall 2: Wenn beide Punkte P_1, P_2 außerhalb der Objekte liegen, so muss auf eine Kollision dazwischen geprüft werden.

Ein Ansatz zur Lösung des zweiten Falls wäre rekursiv das Bewegungssegment zu halbieren und den Mittelpunkt auf Schneidung mit anderem Objekt hin zu überprüfen. Dies ist ein sehr zeiteffizienter Ansatz, der jedoch die Genauigkeit nicht berücksichtigt. Diese Methode empfiehlt sich wenn man einfach nur wissen möchte, ob Kollision auftrat, und es weniger wichtig ist, wo diese genau auftrat.

Ein genauerer Ansatz ist es, die Bewegung als Strahl zu verfolgen und auf Schneiden mit dem BSP-Baum hin zu prüfen (Raytracing). Dies ist ein sehr zeitintensiver Ansatz, der aber die Genauigkeit berücksichtigt, was wiederum zum Vorteil hat, dass die Bewegung, die aus der Kollision resultiert, besser berechnet werden kann.

8. Effizienz von BSP

Eine allgemeine Aufwandsabschätzung für BSP bzw. für das Erstellen von BSP-Bäumen ist nur schwer möglich, denn der Aufwand hängt vom jeweiligen Einsatzgebiet und der gegeben Szene ab. Das Problem hierbei ist die Wahl der Teilungsebenen, durch die viele neue Polygone entstehen können.

Aufwand, der bestenfalls erreicht werden kann:

$$O(n \log(n))$$

Hierbei steht n für die Anzahl der Polygone und $\log(n)$ für die Tiefe des Baumes.

Aufwand im schlechtesten Fall:

$$O(n^2)$$

Durch die Entstehung neuer Polygone bei der Partitionierung kann der Worst-Case Aufwand dies noch übersteigen.

8.1. Verbesserungen der Effizienz

Bounding-Volumina: Es wird eine begrenzende Kugel oder ein Quader um das interessierende Objekt gelegt. Bei einer Kugel wird danach der Kugelmittelpunkt mit der Teilungsebene verglichen. Mit diesem Ansatz kann schnell gesagt werden, wo genau sich ein Objekt befindet. Ein weiterer Vorteil dieser Methode ist die leichte Implementierung einer Bounding-Sphere. Quader haben generell ein besseres Passverhalten als Kugeln und kommen deshalb auch öfter zum Einsatz, beispielsweise in einer *GameEngine.*

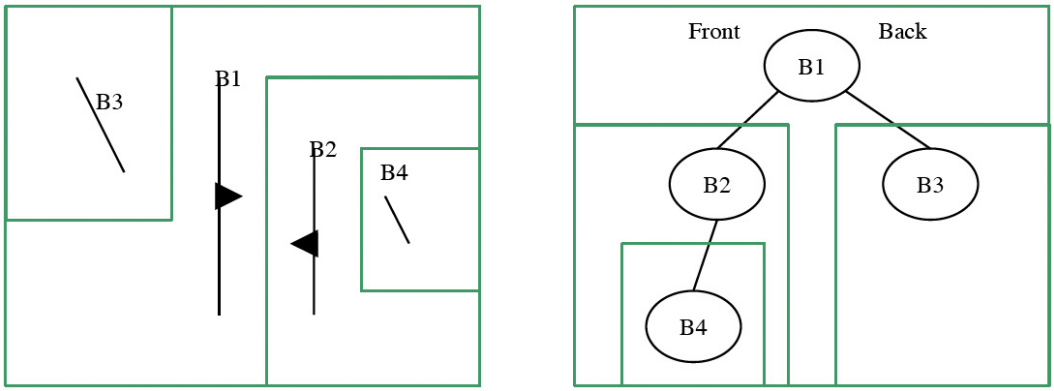


Abbildung 22. Jeder Quader wird unterteilt, bis ein Blatt des BSP-Baumes erreicht ist.

Balancierung des Baumes: Diese Methode kommt vor allem bei Solid-Modelling und Raytracing zum Einsatz. Die Komplexität der Einordnung eines Objekts beruht auf der Tiefe der zugrunde liegenden Baumstruktur. Hiervon ist abhängig, wie lange ein Baumdurchlauf von der Wurzel zu einem beliebigen Blatt dauert. Ein unbalancierter Baum hat in der Regel tiefere Unterbäume, was für Raytracing unvorteilhaft ist. Hingegen ist die Balancierung für HSR belanglos. Die Zeit des Baumdruchlaufs ist bei HSR linear zur Anzahl der vorhandenen Polygone und würde durch eine Balancierung nur gering beeinflusst.

Minimierung der Polygonspaltungen: In der Baumkonstruktionsphase werden die vorhandenen Polygone durch die Teilungsebenen aufgespalten und daraus resultieren neue Polygone. Durch mehr Polygone wird auch mehr Speicher benötigt und es resultieren längere Baumdurchlaufzeiten. Wäre es nun möglich, die Spaltungen zu minimieren, hätte man hiervon ohne Zweifel einen großen Vorteil. Das Problem hierbei ist nur, das Vorkenntnisse über alle beteiligten Polygone existieren müssen, was in der Regel nicht der Fall ist.

9. BSP und Game-Engines

Bei Game-Engines, die technischen Motoren von Videospielen kommt BSP hauptsächlich beim Szenenmanagement zum Einsatz. Für Szenen in Gebäuden oder Bauwerken, speziell für aus Wänden und Korridoren resultierende Geometrien, ist das BSP-Szenenmanagement geeignet, wohingegen BSP für outdoor-artige Szenen nicht geeignet ist.

9.1. Potentially Visible Set

Ein *Potentially Visible Set* (PVS) bezeichnet einen Satz von Polygonen, der von der aktuellen Kameraposition in der Szene potenziell sichtbar ist. Die Idee dahinter ist, dass man Speicher für mehr Geschwindigkeit opfert. Dazu werden die Blätter mit den Polygonen in Cluster gruppiert. In jedem Blattknoten werden *Run Length Encoded*(RLE)-kodiert andere potenziell sichtbare Cluster gespeichert. Wenn das Blatt mit dem Betrachter gefunden wurde, können mit einem Astdurchlauf alle potenziell sichtbaren Polygone ohne Rekursion bestimmt werden und dies mit einem Aufwand von $O(\log(n))$.

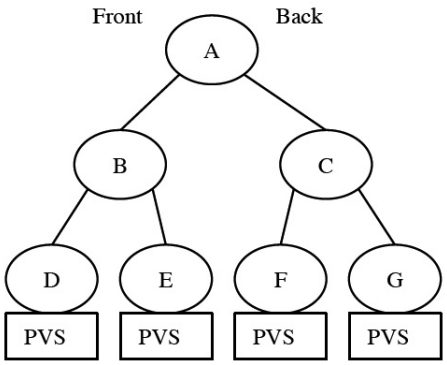


Abbildung 23. Schema Speicherung von PVS

9.2. Portal-based Rendering

Das *Portal-based Rendering* findet Verwendung bei Innenansichten, denn hier kommt nur ein kleiner Bereich für das Rendering in Frage. Die Idee hierbei ist, dass die Umgebung aus Räumen besteht, die durch Fenster und Türen (Portale) miteinander verbunden sind. Hieraus kann man schließen, dass sich der sichtbare Bereich nur auf den aktuellen Raum und angrenzende Räume beschränkt. Das Vorgehen ist folgendermaßen: Man testet die Polygone im Betrachterraum auf Sichtbarkeit und rendert diese gegebenenfalls. Danach folgt ein rekursives Testen auf andere sichtbare Portale und anschließend das Rendern dieser Portale. Ein Vorteil hierbei ist, dass man geometrisch unabhängige Räume verknüpfen kann. *Portal-based Rendering* ist für outdoor-artige Szenen ebenfalls ungeeignet, da keine Portale existieren.

9.2.1. Automatische Portalerzeugung

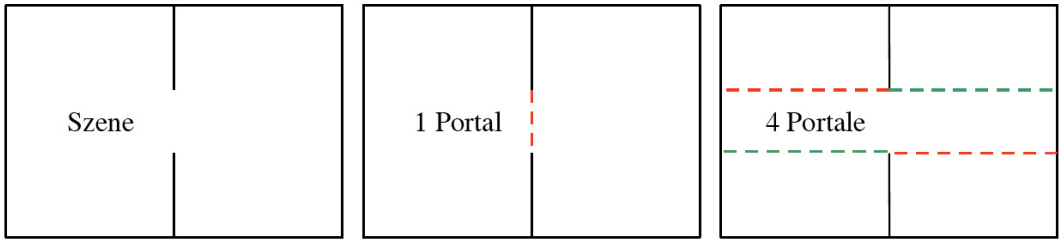


Abbildung 24. Automatische Portalerzeugung

Große Polygone, die an Teilungsebenen ausgerichtet sind eignen sich als Portale, denn Portale sollten immer ein *Front*-Blatt und ein *Back*-Blatt miteinander verbinden. Dies führt jedoch oft zu vielen redundanten Portalen, wie im folgenden Bild verdeutlicht werden soll. Dies macht eine geschickte Reduzierung von Portalen notwendig.

In Abbildung 24 ist zu sehen, wie unterschiedlich die möglichen Portale einer Szene interpretiert werden können. Das Verhältnis von Portalen zu Polygonen hat einen entscheidenden Einfluss auf die Performanz, denn zu viele Portale können schlecht für die Performanz sein.

9.3. Datenstrukturen zur Portalerzeugung

```
class CPortal
{
// Datenstruktur fuer Portal
public:
   Cpolygon  m_Polygon;
   BOOL      m_bCulled;
   int       m_iLeftNode;
   int       m_iRightNode;
}


class CSurperPlane
{
// Datenstruktur fuer Superebene
public:
   void Remove(CPortal *pPortal);
   void Add(CPortal *pPortal);

   TList<CPortal*>    m_LisOfPortals;
}


class CBspNode
{
// Datenstruktur fuer Knoten im BSP-Baum
public:
   int                 m_iTag; //if m_iTag>=0 it is a node else it is a leaf
   CPlane              m_DividingPlane;
   CBspNode           *m_pParent;
   CBspNode           *m_pLeftNode;
   CBspNode           *m_pRightNode;
   Tlist<CTriangle>    m_ListOfTriangle;
}


class CBspTree
{
public:
   // Haupt-Funktion des Algorithmus
   void GeneratePortal(CBspTreeNode *pBspNode_, CSuperPlane *pSuperPlane_);

   // Teste, ob das Dreieck sich mit dem Polygon schneidet
   BOOL TriangleIntersectPolygon(CTriangle& Tri_,CPolygon& Polygon_);

   // Pruefe Beziehung von Teilungsebene zu Polygon
   // Entweder ist Polygon auf positiver
   // oder negativer Seite der Teilungsebene
   int CalculateSide(CPlane plane_,CPolygon *pPolygon_);


   // Benutze Teilungsebene, um das Portal zu clippen und weise das Resultat 'leftportal'
   // und ,rightportal' zu. 'leftportal' = positive Seite, 'rightportal' = neg. Seite
   void ClipPortal(CPlane plane_, CPortoal *pPortal_,
                   CPortoal& PortalLeft_, CPortoal& PortalRight_);

   CBspNode              *m_pRoot;
   TList<CSuperPlane*>    m_ListOfSuperPlanes;
}
```

9.4. Algorithmus zur Portalerzeugung (Pseudocode)

```
void CBspTree::GeneratePortal(CBspTreeNode *pBspNode_, CSuperPlane *pSuperPlane_)
{
   if (pBspNode_->IsLeaf ())
   {
        1. Benutze jedes Dreieck in pBspNode_ fuer Clipping gegen jedes Portal in
      pSuperPlane_;
        2. Aktualisiere die Validierungsinforamtion fuer jedes Portal
        3. Loesche alle Portale die pBspNode_ in pSuperPlane_ verbinden;
   }
   else
   {
      if (pSuperPlane_)
      {
        1. Benutze Teilungsebene von pBspNode_ fuer Clipping gegen jedes Portal in
      pSuperPlane_;
        2. Aktualisiere Listen von Verbindungen fuer jedes Portal;
        3. GeneratePortal(pBspNode_->m_pLeftNode,pSuperPlane_);
        4. GeneratePortal(pBspNode_->m_pRightNode,pSuperPlane_);
      }

      if (noch keine Superebene in pBspNode_ generiert wurde)
      {
        1. Erzeuge neue pNewSuperPlane auf der Teilungsebene von pBspNode_
        2. Erzeuge ein großes Portal(Polygon) auf pNewSuperPlane;
        3. Clippe Portal auf pNewSuperPlane, um sicher zu gehen, dass es innerhalb
      des Raumes von pBspNode_
        4. GeneratePortal(pBspNode_->m_pLeftNode,pNewSuperPlane);
        5. GeneratePortal(pBspNode_->m_pRightNode,pNewSuperPlane);
      }
   }
}
```

Literatur

[1] BSP-FAQ, http://www.opengl.org//resources/code/samples/bspfaq/

[2] iX Magazin, *MGame Engines*, S.52 ff, September 2006

[3] Wikipedia, http://de.wikipedia.org/wiki/Binary_Space_Partitioning

[4] Gamedev, http://www.gamedev.net/reference/programming/features/apg/

Teil III

Filmkodierung

Stefan Zimmer[4]

Abstract. *In diesem Artikel werden wesentliche Filmkodierungstechniken bzw. Filmkompressionstechniken besprochen. Ausgehend von der zweidimensionalen JPEG-Kompression mit der* Discrete Cosine Transform *(DCT) und der* Huffman-*Entropiekodierung werden später die Ideen der MPEG-Standards besprochen, zu denen im weiteren Sinne auch DivX und H.264 gehören,.*

1. Einleitung

Seit Anfang dieses Jahrhunderts werden analoge Filmspeichermedien, wie die Videokassette, mehr und mehr durch digitale Speichermedien, wie CD-Roms oder DVDs, ersetzt. Daneben wird auch das WorldWideWeb immer multimedialer: Filmtrailer oder ganze Filme können heruntergeladen werden, Fernsehsender verbreiten ihr Programm auch über das Internet. Sogar das terrestrische Fernsehen setzt immer mehr auf digitale statt analoge Übertragung. Nicht nur die Medienbranche profitiert von der digitalen Filmverarbeitung, auch wissenschaftliche oder medizinische Anwender können neuste Erkenntnisse von fernen Satelliten, Vorlesungen oder Diagnosen auf digitalem Wege schnell, sicher und effizient verbreiten. In einer globalisierten und stark vernetzten Welt werden vermehrt Filmkodierungstechniken eingesetzt. Man denke nur an Videokonferenzen oder an Überwachungen von Maschinen an anderen Orten.

Die größte Herausforderung ist hierbei die Reduktion der Datenmenge, ohne dabei die Qualität des übertragenen Videostreams merklich zu beeinträchtigen. Eine kleine Rechnung soll dies hier verdeutlichen. Ein Fernsehbild hat ca. 400 auf 600 Pixel. Jedes Pixel kann man durch drei Farbwerte darstellen: Rot, Grün, Blau. Wenn man für jeden Farbton 256 Abstufungen haben möchte (dies entspricht einem durchschnittlichen Monitor), dann benötigt man 720 kByte pro Bild. Der PAL Standard in Europa sieht 50 Halbbilder, also 25 Ganzbilder, in der Sekunde vor. Multipliziert man beide Werte bekommt man einen Datenstrom von ca. 144 MBit pro Sekunde. Zum Vergleich: ein DSL Anschluss hat 2 MBit/s, ein USB 2.0 Anschluss ca. 480 MBit/s und ein Modem 0.056 MBit/s, eine CD Rom wäre nach 38 Sekunden gefüllt, eine DVD nach ca. 10 Minuten. Bei neuen Standards wie HD TV mit höheren Auflösungen und größerer Bildwiederholungsrate werden die Datenmengen noch viel größer.

Solch große Datenmengen können auch mit heutiger Computer- und Netzwerkhardware nur sinnvoll in komprimierter Form gehandhabt werden. Hierbei werden drei verschiedene Ansätze verfolgt. Der erste Ansatz versucht Redundanzen im Videostream zu minimieren. Dies kann sowohl in der räumlichen, als auch in der zeitlichen Dimension passieren. Wenn ein Film über mehrere Einzelbilder

[4]IWR, Universiät Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Stefan_Zimmer@gmx.de

hinweg den gleichen Hintergrund oder ähnlichen Hintergrund besitzt, reicht es, die Unterschiede zu speichern. Dies könnte beispielsweise ein blauer Himmel sein.

Der zweite Ansatz nutzt Eigenschaften des menschlichen Auges aus und möchte so unwesentliche Informationen aus dem Film filtern. Hier wird verlustbehaftet gearbeitet. In einer Landschaft werden beispielsweise hauptsächlich die groben Umrisse wahrgenommen und weniger stark feine Strukturen, wie einzelne Grashalme oder die Rillen auf einem Blatt eines Baumes. Auf diese mangelnde Wahrnehmung wird mit Hilfe der DCT und Quantisierung reagiert. Das menschliche Auge kann aufgrund der Nachtsehzäpfchen auch viel mehr Helligkeitstöne, als Farbtöne auflösen. Hiermit wird die YUV Bildkodierung motiviert, die die unterschiedlichen Farbkanäle mit verschiedener Präzision kodiert.

Der dritte Ansatz nutzt die in der Informatik bekannten Entropiekodierungsverfahren. Um 1000 Nullen darzustellen, kann ich entweder 1000 Nullen auf ein Blatt Papier schreiben, oder aber den Satz „Auf dem Blatt stehen 1000 Nullen“, der wesentlich kürzer ist.

Im Folgenden werde nun die einzelnen Komprimierungsverfahren vorgestellt, sowie deren Integration in gängige Medienformate wie JPEG oder MPEG.

2. Struktur-Kodierung

2.1. YUV Kodierung

Das menschliche Auge ist auf Helligkeit viel empfindlicher als auf Farben. Darüber hinaus tragen verschiedene Farben auch unterschiedlich stark zu unserem Bildempfinden bei, dass heißt sie werden in ihrer Helligkeit unterschiedlich stark wahrgenommen. Nach einigen Tests fand man heraus, dass rot etwa drei mal so stark wie blau, grün dagegen etwa doppelt so stark wie rot wahrgenommen wird. Man geht deshalb von der Computergraphik üblichen RGB (Rot, Grün, Blau) Darstellung zur YUV (Gesamthelligkeit und zwei Farbkanänle) Darstellung über. Diese Transformation kann mit Hilfe einer invertierbaren Matrix

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

oder als inverse Transformation

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

geschrieben werden. Bis hierhin wurde lediglich eine Transformation durchgeführt. Der eigentliche Trick, der zu einer Kompression der Datenmenge führt, ist die unterschiedlich gute Kodierung der einzelnen YUV Kanäle. Beim analogen Fernsehen wird beispielsweise der Y-Kanal mit 6 MHz, der V-Kanal mit 1.5 MHz und der U-Kanal, aus dem im Dekodierungsprozess das relativ unwichtige Blau extrahiert wird, nur mit 0.6 MHz übertragen. In der digitalen Filmkodierung steuert man die Auflösung nicht über die Frequenz, sondern über die unterschiedliche Anzahl von Pixel, die den einzelnen Kanälen in jedem Frame zugeordnet wird. Ich werde im Abschnitt über MPEG-Standards näher darauf eingehen.

2.1.1. Ein praktisches Beispiel

In folgenden Bildern kann man die YUV Transformation nachvollziehen. Das Originalbild wird als normales RGB bzw. YUV Bild dargestellt (die Ergebnisse von RGB und YUV sollten bei entsprechender Monitorkalibrierung identisch sein). Darunter ist der Y-Kanal, der die Helligkeitsverteilung im Bild kodiert. Dieser entspricht einer Aufnahme mit einem schwarz-weißFilm. Wie im Y-Kanal entsprechen auch im U- und V-Kanal weiße Pixel großen Intensitäten und schwarze Pixel geringen Intensitäten. Vergleicht man das Originalbild mit dem U-Kanal (z.B. im Himmel), wird deutlich, dass in diesem Kanal hauptsächlich die Blautöne gespeichert werden. Im V-Kanal werden hingegen die Rottöne gespeichert (siehe das rötliche Holz der Hütte). Dies erkennt man auch an der inversen YUV Transformation. Neben einem Helligkeitsoffset wird der R-Kanal von der V-Komponente und der B-Kanal von der U-Komponente gesteuert.

Abbildung 25. Originalbild einer Berglandschaft mit allen Kanälen

Abbildung 26. Helligkeits- bzw. Y-Kanal

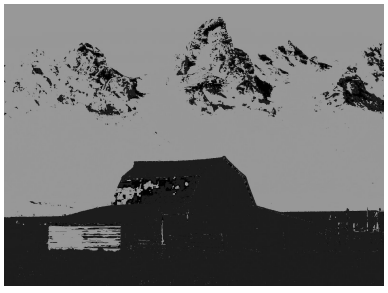


Abbildung 27. Erster Farb- bzw. U-Kanal

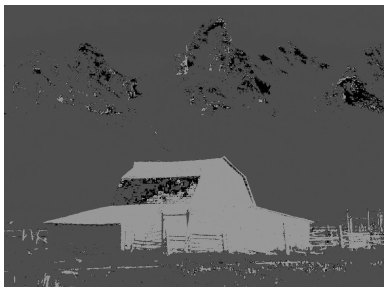


Abbildung 28. Zweiter Farb- bzw. V-Kanal

2.2. Discrete Cosine Transform (DCT)

Die Idee der DCT ist ähnlich zu der, die bei mp3-Musikdateien verwendet wird. Hier wird ein Musikstück in zeitlich kurze Teilstücke zerlegt und in den Frequenzraum fouriertransformiert. Im Frequenzraum reicht es oft, wenn man die fünf größten Frequenzen speichert, da das menschliche Ohr meist nicht mehr Töne gleichzeitig erkennen kann. Die Idee ist also: Erkenne die Struktur in kleinen Teilabschnitten und speichere die wichtigsten Strukturelemente. Beim Abspielen wird die fourierkodierte Musikdatei wieder rücktransformiert und auf herkömmliche Weise abgespielt.

Eine ähnliche Idee verfolgt man bei der DCT, mit der man Strukturen in Bildern erkennen möchte. Die Komprimierung erfolgt erst in einem zweiten Schritt, der Quantisierung, und wird weiter unten beschrieben.

Für die DCT wird jeder Kanal (z.B. YUV) des Bildes in 8x8 Pixel große Quadrate aufgeteilt. Danach wird für jedes Quadrat eine eigene Transformation durchgeführt. Die Frequenzen F_{uv} des transformierten Bildes ergeben sich aus den ursprünglichen Intensitäten I_{xy} mit folgender Formel:

$$F_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} I_{xy} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

$$\text{wobei} \quad C_u, C_v = \frac{1}{\sqrt{2}} \ \text{ für } \ x, v = 0; \quad C_u, C_v = 1 \quad \text{sonst.}$$

Die Rücktransformation (IDCT) geschieht mit den gleichen Koeffizienten und der Formel:

$$I_{uv} = \frac{1}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} F_{xy} C_x C_y \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}.$$

Durch die einzelnen Koeffizienten werden verschiedene Muster erzeugt, die überlagert das komplette Bild ergeben. In der Abbildung 29 sind die einzelnen Basisfunktionen dargestellt. Der linke obere Koeffizient heißt DC Koeffizient, da er den gleichmäßigen Hintergrund darstellt. Alle anderen Koeffizienten heißen AC Koeffizienten. Sie stellen immer ein wellenartiges Muster dar.

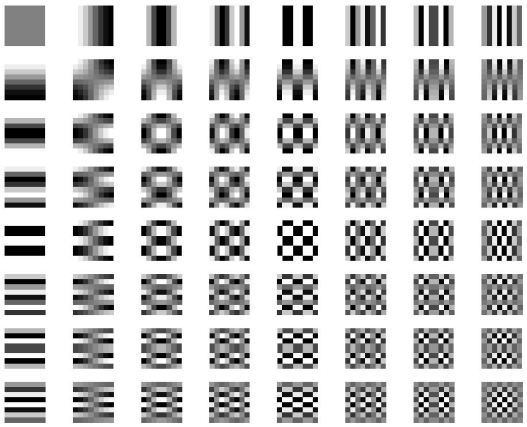


Abbildung 29. Basisfunktionen der DCT

2.2.1. Ein praktisches Beispiel

Folgende Bilder zeigen welche Bildinformationen die hohen und welche die niedrigen Frequenzen tragen. Wenn bei der DCT die hohen Frequenzen berechnet werden, hat der Kosinus in der Transformationsformel eine hohe Frequenz. Bei der Bestimmung der DCT-Koeffizienten für die hohen Frequenzen von großen homogenen Flächen wird dementsprechend der gleiche Farbwert der Fläche häufig mit unterschiedlichen Vorzeichen addiert, da der Cosinus zwischen -1 und 1 schwingt. Im Mittel, das heißt wenn alle Pixel aufaddiert wurden, ergibt dies ungefähr null. Deshalb ist die gleichmäßig weiße Mütze im hochfrequenten Bild beinahe verschwunden, sie wird fast schwarz dargestellt. Mit den groben Basisfunktionen der niedrigen Frequenzen können auf der anderen Seite keine feinen Strukturen aufgelöst werden. Die Knöpfe am Pullover sowie das Gitter im Hintergrund sind im niederfrequenten Bild nicht zu erkennen.

Abbildung 30. Originalbild einer Person

2.3. Kompression mit Hilfe der DCT

Die Kompression des DCT transformierten Bildes wird durch zweierlei Maßnahmen erreicht:

Abbildung 31. Niederfrequenter Anteil nach DCT

Abbildung 32. Hochfrequenter Anteil nach DCT

Die erste Maßnahme besteht darin, die einzelnen Koeffizienten der DCT Transformation nicht alle mit der gleichen Anzahl von Bits abzuspeichern. Hier können unterschiedliche so genannte Quantisierungsmatrizen verwendet werden. Für Zeichnungen und Text sind die hohen Frequenzen wichtiger, dementsprechend sollten die hohen Frequenzen hier auch mit mehreren Bits abgespeichert werden. Es macht für die Lesbarkeit eines Textes nichts aus, ob der Hintergrund hellblau oder dunkelblau ist, die Kanten der einzelnen Buchstaben sollten jedoch klar erkennbar sein und keinesfalls ineinander übergehen. Bei natürlichen Bildern, wie sie in Filmen vorkommen, ist es wichtig, große Objekte wie Personen oder Autos zu erkennen. Feine Strukturen wie beispielsweise eine Wiese oder die Steine auf dem Boden sind, um den Informationsgehalt des Bildes zu erfassen, nicht so wichtig. Dementsprechend werden in solchen Fällen die Koeffizienten der niedrigen Frequenzen genauer abgespeichert als die der hohen. Mit der Gesamtzahl der zur Kodierung der Frequenzen verwendeten Bits kann man die Kompressionsrate einstellen.

Praktisch wird der quantisierte DCT Koeffizient F^q_{uv} aus dem ursprünlichen Koeffizient F_{uv} und dem Eintrag q_{uv} in der Quantisierungsmatrix nach folgender Formel berechnet:

$$F^q_{uv} = integer(round(F_{qv}/q_{uv})).$$

Die zweite Maßnahme wird durch die DCT vorbereitet. Wenn man sich das Ergebnis einer DCT eines typischen Bildausschnitts anschaut, stellt man fest, dass in der linken oberen Ecke (d.h. bei den niedrigen Frequenzen) ein unregelmäßiges Muster entsteht. Der Rest (die höheren Frequenzen) sind hingegen ähnlich hoch und bilden meist ein regelmäßiges Muster oder gar ein Plateau. Mit einer gröberen Quantisierung entstehen immer mehr ähnliche Muster. Diese Muster können mit Entropie-Kodierungsverfahren, wie sie weiter unten beschrieben werden, sehr kompakt gespeichert werden. Damit möglichst lange Muster erkannt werden, hat es sich empirisch (da in der oberen linken Ecke im Frequenzraum die meiste „Unordnung“ ist) als günstig herausgestellt, nicht Zeile für Zeile abzuspeichern, sondern in einem Zickzack-Schema vorzugehen.

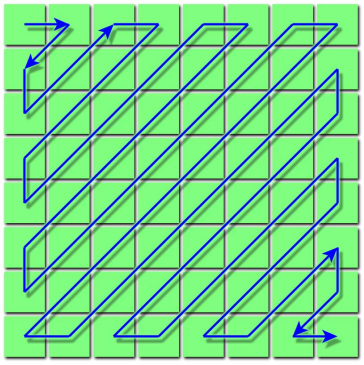


Abbildung 33. Zickzack-Schema beim Abspeichern der DCT Koeffizienten

2.4. JPEG-Standard

In JPEG-Bildern, wie sie im Internet weit verbreitet sind, finden sich die oben genannten Komprimierungsmaßnahmen wieder. In den Abbildungen 34 und 35 sind typische Encoder und Decoder für JPEG-Bilder dargestellt. Der Encoder transformiert das Bild zuerst mit der FDCT (forward DCT). Danach kann eine optimale Quantisierungsmatrix definiert werden. Sie kann frei gewählt werden, da sie im komprimierten Bild mit übertragen wird. Je nach dem wie die Quantisierungsmatrix gewählt wird, kann so zwischen einem qualitativ guten Bild mit großem Speicherplatzbedarf, und einem weniger guten mit kleinerem Platzbedarf unterschieden werden. Es kann für Text oder Bildinformationen optimiert werden. Die quantisierten Koeffizienten eines jeden 8x8 Blocks werden auch für die folgende Entropiekodierung separat betrachtet. Jeder Block wird im Zickzack-Verfahren zuerst einer Lauflägenkodierung unterzogen und später mit einer Huffman-Kodierung optimiert. Die für die Huffman-Kodierung nötige Tabelle wird ebenfalls übertragen. Die Entropiekodierungen werden weiter unten genauer beschrieben. Der Decoder invertiert dieses Prinzip.

2.4.1. Ein praktisches Beispiel

Das nachfolgende Beispiel (Abbildungen 36 und 37) zeigt deutlich den Einfluss des Quantisierungsfaktors auf das Ergebnis. Im Originalbild sind die feinen Strukturen (Blätter und Stängel der Pflanzen) gut zu erkennen. Des Weiteren ist das Bild homogen, das heißt alle Farbübergänge sind glatt. Im quantisierten Bild erkennt man die Quadrate, in die das Bild für die DCT eingeteilt wurde, da die DC

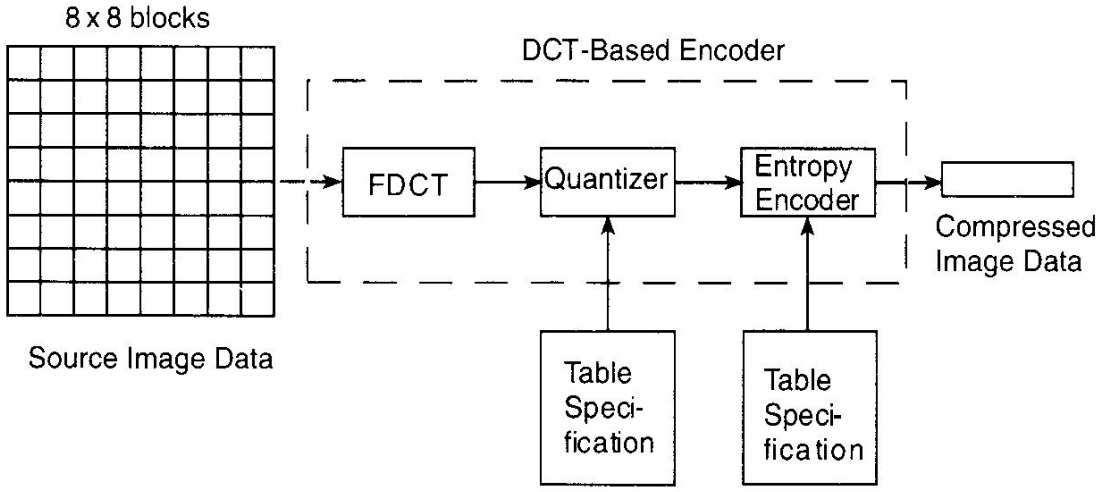


Abbildung 34. Typischer Aufbau eines JPEG-Encoders

Koeffizienten von benachbarten Quadraten aufgrund der Quantisierung immer einen relativ großen Unterschied haben müssen, oder aber gleich sind. Ähnliches erkennt man auch für die hohen Frequenzen. Verfolgt man einen Pflanzenstängel über eine Quadratgrenze hinweg, erkennt man, dass er nicht nahtlos anschließt, sondern einen gewissen Sprung macht. Ebenso erkennt man in den Blättern Quadrate mit unterschiedlichen Grüntönen. Dies rührt von der Quantisierung des DC Koeffizienten her.

3. Entropie-Kodierung

3.1. Definition der Entropie

Ziel jeglicher Entropie-Kodierung ist es, gegebene Daten verlustfrei mit minimalem Speicherplatzbedarf abzuspeichern. Die Entropie stellt hierbei die theoretische Grenze dar. Sie ist das Maß für den Informationsgehalt von Daten. Je größer die Entropie, desto größer der Informationsgehalt und desto kleiner die mögliche Kompression.

$$E = -\sum_{i=0}^{N} p(x_i) log_2(\frac{1}{p(x_i)})$$

Hierbei wird die Entropie einer Datenquelle mit einem Alphabet von N Buchstaben berechnet. $p(x_i)$ ist die relative Häufigkeit des Buchstabens x_i. Der Wert der Entropie gibt an, wie viele Bits bei optimaler Kodierung im Schnitt für ein Zeichen verwendet werden müssen.

3.2. Lauflängenkodierung

Die Lauflängenkodierung ist die einfachste Form der Entropiekodierung. Hier werden gleiche aufeinanderfolgende Zeichen durch ein Zeichen und eine Anzahl ersetzt. Zusätzlich wird ein Escape-Zeichen eingeführt. Es signalisiert beim Dekodieren, dass das übernächste Zeichen die Anzahl x und das nächste Zeichen dasjenige ist, welches x mal wiederholt ausgegeben werden soll. Ist das Escape-

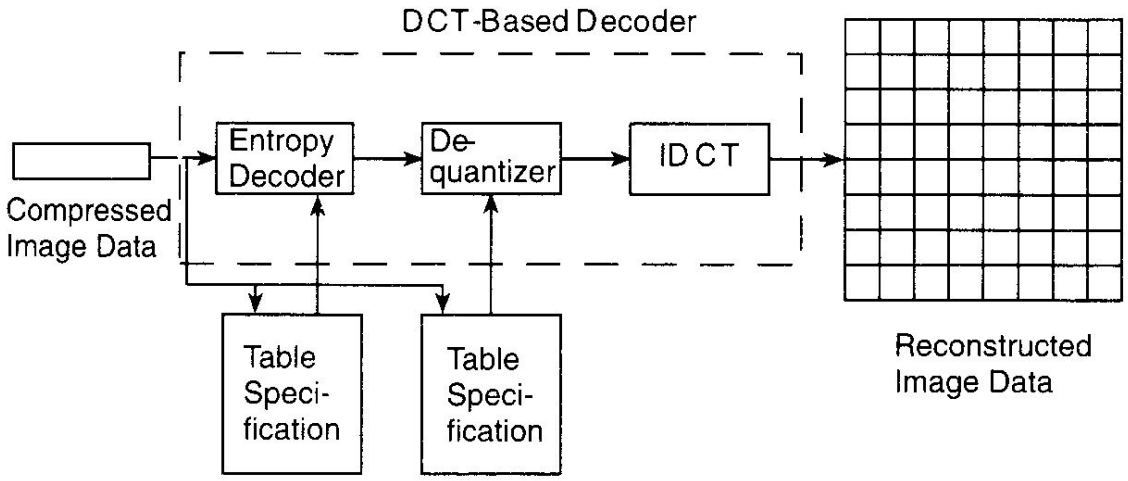


Abbildung 35. Typischer Aufbau eines JPEG-Decoders

Abbildung 36. Originalbild einer Wiese

Zeichen selbst Teil der ursprünlichen Daten, wird es beim Encodieren durch zwei Escape-Zeichen ersetzt. Der Decoder ersetzt diese wieder durch ein Escape-Zeichen.

3.3. Huffman-Kodierung

Die Huffman-Kodierung wird für eine vor der Kompression bekannte Datenmenge eingesetzt. Es können also nicht unbekannte Quellen kodiert werden. Die Idee hierbei ist, ähnlich wie beim Morsealphabet, für häufig auftretende Buchstaben kurze und für seltener auftretende Buchstaben lange Bitsequenzen zu benutzen. Somit wird im Sinne der Entropie eine auf ein Bit pro Buchstaben optimale Kompression erreicht. Der Algorithmus lässt sich in folgenden Schritten zusammenfassen:

- Die Wahrscheinlichkeit für jeden einzelnen Buchstaben wird aus seiner relativen Häufigkeit bestimmt.
- Es wird ein Kodierungsbaum erstellt (vgl. Abbildung 39): Man startet mit einem Knoten für

Abbildung 37. Mit dem Quantisierungsfaktor 12 kodiertes Bild

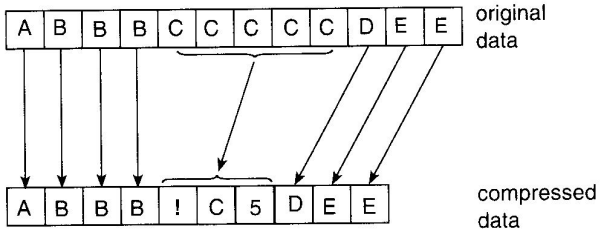


Abbildung 38. Beispiel einer Lauflängenkodierung

jeden Buchstaben des Alphabets. Jeder Knoten erhält als Gewicht die Wahrscheinlichkeit des zugehörigen Buchstabens.

- Die zwei Knoten mit dem geringsten Gewicht werden zusammengefasst zu einem neuen Knoten. Dieser erhält als Gewicht die Summe der Gewichte der zusammengefassten Knoten. In kommenden Schritten werden die alten, d.h. zusammengefassten Knoten, nicht mehr betrachtet, sondern nur noch der neue mit dem neuen Gewicht.
- Wie im letzten Schritt werden die jeweils kleinsten Knoten so lange zusammengefasst, bis man einen Wurzelknoten mit dem Gewicht 100 Prozent erhält. Bei gleichen Gewichten wird derjenige Knoten bevorzugt, der weniger Tochterknoten hat. Da es nicht immer eindeutig ist, welche Knoten zusammengefasst werden, ist auch die gesamte Huffman-Kodierung nicht eindeutig.
- An jedem Knoten wird jeweils der linke/obere Tochterknoten mit der Ziffer 1 und der rechte/untere mit der Ziffer 0 bezeichnet.
- Im letzten Schritt wird die Lookup-Table, das heißt die Zuordnung von Bitfolgen zu den Buchstaben des Alphabets erzeugt. Hierzu läuft man ausgehend vom Wurzelknoten den Weg zu jedem einzelnen Buchstaben ab. Die Ziffern ergeben in der Reihenfolge, wie man sie auf dem Weg antrifft, den Code für die einzelnen Buchstaben.

Das praktische an dieser Art der Kodierung erkennt man bei der Decodierung. Wenn der Decoder die Lookup-Table und einen Bitstrom von Huffman-kodierten Zeichen erhält, kann er diese dekodie-

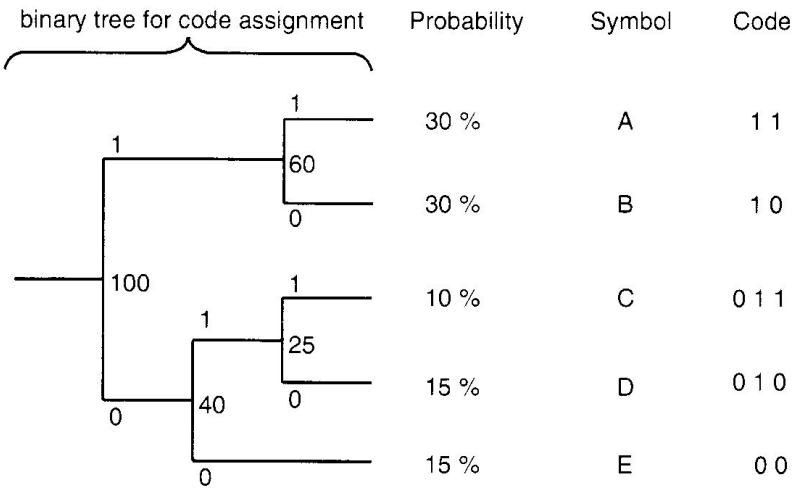


Abbildung 39. Beispiel einer Huffman-Kodierung

ren. Insbesondere müssen zwischen den einzelnen Zeichen keine Trennzeichen eingefügt werden, da kein Buchstabencode am Anfang einen anderen kürzeren Buchstabencode enthält. Somit kann der Datenstrom eindeutig dekodiert werden.

3.4. Arithmetische Kodierung

Die Huffman-Kodierung hat bei wenigen Buchstaben, bei denen einer mit einer Wahrscheinlichkeit größer als 0.5 vorkommt, eine im Vergleich zur Entropie schlechte Kompression. Ein Beispiel soll dies verdeutlichen. Besteht das Alphabet aus 0 und 1, dann wird bei der Huffman-Kodierung jeder Buchstabe mit genau einem Bit codiert. Im Schnitt ergibt dies 1 Bit pro Buchstaben. Nach der Entropieformel wäre aber bei $p(1) = 0.2$, $p(0) = 0.8$ $E = 0.2 \log(\frac{1}{0.2}) + 0.8 \log(\frac{1}{0.8}) = 0.72$ ein durchschnittlicher Bitverbrauch von 0.72 Bit pro Zeichen möglich. Es werden also 28 Prozent der möglichen Kompression verschenkt. Bei Bildern sind diese Wahrscheinlichkeiten durch die DCT und Quantisierung durchaus möglich. Die *arithmetische Kodierung* hat dieses Problem nicht. Sie ist nicht wie die Huffman-Kodierung auf das nächste Bit entropieoptimal, sondern kodiert mit genau dem Wert, den die Entropie erlaubt.

Bei der Kodierung wird das Intervall entsprechend der Antreffwahrscheinlichkeiten der einzelnen Buchstaben unterteilt (vgl. Abbildung 40). Jedem Subintervall wird sein Buchstaben zugeordnet. Das Subintervall, das dem ersten Buchstaben der Zeichenfolge entspricht, wird wieder entsprechend der Antreffwahrscheinlichkeiten der Buchstaben unterteilt. Dies wird rekursiv wiederholt, bis das Intervall für den letzten Buchstaben der Zeichenkette gefunden ist. Die gesamte Zeichenkette wird nun durch eine beliebige Zahl aus dem Intervall des letzten Buchstabens repräsentiert. Nach gewissen Theoremen gibt es in diesem Intervall immer eine Zahl, deren Kodierungslänge nicht länger als die minimal nötige Kodierungslänge für die Zeichenfolge ist. Eine Zeichenfolge, in der die Buchstaben entsprechend der in der Entropieberechung eingesetzten Wahrscheinlichkeiten vorkommen, wird also exakt mit *Entropie* mal *Anzahl Buchstaben Bit* kodiert und nicht mit *RoundUptofullBit (Entropie)* mal *Buchstaben* wie bei der Huffman-Kodierung.

Beim Dekodieren wird in umgekehrter Reihenfolge vorgegangen. Das Intervall zwischen Null und

Eins wird so lange unterteilt, bis man das zur empfangenen Zahl gehörige Intervall gefunden hat. Hierfür müssen neben der Zahl, die die Zeichenkette kodiert, auch die Wahrscheinlichkeiten der einzelnen Buchstaben bekannt sein. Dies kann im Voraus festgelegt oder aber mit übertragen werden.

Um das Ende einer Zeichenfolge zu erkennen muss entweder ein Terminalzeichen eingeführt werden, oder aber im Voraus bekannt sein, wie lang das Paket ist. Dies kann beispielsweise durch einen im Standard definierte Packetgröße erreicht werden. Z.B. immer 188 Byte gehören zusammen und werden als eine Zahl kodiert.

In dem in Abbildung 40 dargestellten Beispiel wird beispielsweise die Zeichenkette „ACB“ durch die Zahl 0.14, aber auch durch 0.125 kodiert. In einem binären Zahlensystem wird man 0.125 wählen, da dies eine am kürzesten zu kodierende Zahl (0.125 dezimal = 0.01 binär) aus dem Intervall ist. Es reicht ”01ßu übertragen, da jede Zahl mit „0.“ anfängt und dieses Präfix somit automatisch davor geschrieben werden kann.

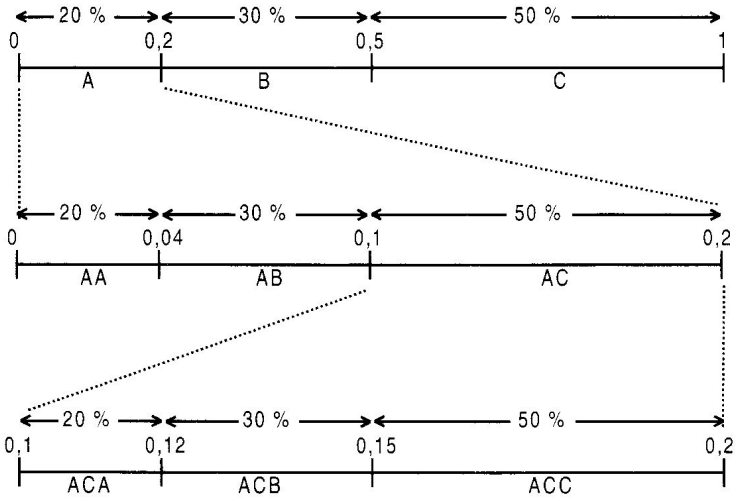


Abbildung 40. Beispiel einer arithmetischen Kodierung

4. Raum-Zeit-Kodierung

4.1. Einleitung in die Standards

In der vorangegangenen Kapiteln wurde beschrieben, wie man nicht bewegte Bilder möglichst effizient speichert. Einen Film könnte man mit dieser Technik ähnlich der alten Super 8 Technik speichern. Frame für Frame JPEG-kodiert auf einem Datenträger. Manche Videokameras verfahren heute so, da sie für eine weitergehende Echtzeitkompression nicht genug Speicher und Rechenleistung haben. Der Nachteil hierbei ist, dass so nur 30 Minuten Film auf eine spezielle wiederbeschreibbare DVD passen. Um Filme auf CD Roms zu speichern, Filmtrailer über das Internet zu verbreiten oder für eine Videokonferenz über die ISDN-Leitung oder das Handy sind diese Datenmengen eindeutig zu groß und müssen deshalb weiter komprimiert werden.

Glücklicherweise enthalten die einzelnen Bilder eines Filmes große Redundanzen. Der Hintergrund bleibt oft über mehrere Frames hinweg erhalten. Selbst bei einem Kameraschwenk bestehen große

Teile des neuen Frames aus dem alten Frame, lediglich die Position einer ganzen Pixelgruppe hat sich verändert. Gleiches geschieht, wenn sich in einer Szene eine Person oder ein Gegenstand bewegt. In allen diesen Fällen geht es darum, gleiche oder ähnliche Bildausschnitte zu finden und diese Ähnlichkeiten bei der Kodierung auszunutzen. Hierzu wird der eigentliche Bildausschnitt nur ein oder zwei Mal als Bild gespeichert. In den restlichen Frames, in denen der Bildausschnitt vorkommt, wird nur ein Bewegungsvektor und ein Differenzbild gespeichert. Der Bewegungsvektor gibt an, wo der Bildausschnitt im vorangegangenen Frame steht. Im Differenzbild werden die nötigen Korrekturen gespeichert. Alternativ kann ein Frame aus einem vorigen und einem nachfolgenden Frame durch Interpolation berechnet werden. Im MPEG-Standard wird dies mit so genannten P-, I-, B- und D-Frames implementiert. Dies wird weiter unten genauer beschrieben. Ein möglicher Algorithmus für das Auffinden von ähnlichen Bildausschnitten wird ebenfalls vorgestellt. Für diesen Algorithmus sieht der MPEG-Standard keine prinzipiellen Einschränkungen vor, lediglich der Umkreis, aus dem die ähnlichen Bildausschnitte kommen dürfen, ist manchmal eingeschränkt, um die Implementierung in Hardware zu erleichtern und um die Encodierung zu beschleunigen.

Eine weitere Möglichkeit der Kompression ergibt sich besonders bei Animationen. Hier kann man ganze Objekte durch eine Objektbeschreibungssprache wie z.B. Flash oder MIDI darstellen. Ich werde darauf kurz im Kapitel über Objektorientiertheit in MPEG-4 eingehen.

4.2. Suchalgorithmus für ähnliche Bildausschnitte

Um ähnliche Bildausschnitte zu identifizieren, kann man folgenden Algorithmus verwenden. Der 16x16 Pixel große Makroblock mit den Pixeln $r_{l,m}$ $l = 0..15, m = 0..15$ soll mit dem 16x16 Pixel großen Makroblock $u_{x,y}$ $x = 0..15, y = 0..15$ in einem vorangegangenen Frame verglichen werden. In wie weit die zwei Makroblöcke u und v übereinstimmen, kann mit der Norm

$$d(u,v) = \sqrt{\sum_{x,y=0..15} (u(x,y) - r(x,y))^2}$$

festgestellt werden. Je nach dem wie der Makroblock u definiert wird, kann so ein gesamter vorausgehender Frame abgesucht werden. Wenn die Norm einen kritischen Wert unterschreitet, ist ein ähnlicher Block gefunden. Je nach dem wie groß dieser Wert gewählt wird, ist dieses Suchverfahren mehr oder weniger tolerant bei der Gleichheitsbeurteilung von zwei Makroblöcken. Um das Verfahren zu beschleunigen, kann man sich bei der Suche zunächst auf den Y-Kanal beschränken.

Prinzipiell entscheidet der Suchalgorithmus über die Qualität und vor allem die Geschwindigkeit eines Encoders. Neben einfachen Ansätzen, ähnlich dem oben vorgestellten, könnten hierzu auch neuronale Netze eingesetzt werden, die auf den zu suchenden Makroblock trainiert werden und dann im anderen Frame suchen. Dies ist jedoch ein unübliches Verfahren. Auch statistische Methoden, wie sie in [6] vorgeschlagen werden, könnten adaptiert werden. Oder es kann für eine erste schnelle Analyse nur der Mittelwert und die Varianz der Makroblöcke verglichen werden. Diese müssen nur einmal für jeden Makroblock im vorangegangenen Frame berechnet werden, da sie nicht vom Vergleichsmakroblock abhänig sind. Eine andere Idee wäre, Kantenfilter zu benutzen und zunächst nur die Kanten zu vergleichen. Um das Feature (wird weiter unten vorgestellt) der beliebig geformten Objekte in MPEG-4 auszunutzen, muss man erst einmal erkennen, welche Pixel zu einem Objekt gehören. Meiner Meinung nach könnte man in mehreren aufeinanderfolgenden Frames nach Bewegungsvektoren suchen. Alle Pixel, auch in verschiedenen Frames, mit den gleichen oder ähnlichen Bewegungsvektoren werden zu einem Objekt zusammengefasst. Eventuell muss man hierbei Rotationen etc. herausrechnen oder tolerieren.

Besonders für die Kodierung mit variabler Bitrate, wie bei DivX, benötigt man zudem einen Algorithmus, der erkennt, wie viel Bewegung in einem Frame ist. Hierzu kann man beispielsweise die durchschnittliche Veränderung jedes Pixel ausrechnen. Zeitlich aufeinanderfolgende Frames mit einer hohen Veränderung werden zu einem Objekt zusammengefasst und qualitativ schlechter kodiert.

In jedem Fall benötigt die Suche nach ähnlichen Makroblöcken immer sehr viel Rechenzeit.

4.3. P-, I-, B-, D-Frames in den MPEG-Standards

Wie bereits oben erwähnt wird in MPEG-Standards die zeitliche Redundanz in einem Film durch P-, I- und B-Frames reduziert. D-Frames sind hierzu eigentlich nicht nötig und werden nur der Vollständigkeit halber erwähnt (D-Frames sind I-Frames, bei denen nur der DC-Koeffizient gespeichert wird, sie können beim schnellen Lesen des Filmes z.B. beim Spulen nützlich sein). In MPEG-Standards werden mehrere Frames zu einer *Group of Pictures* (GOP) zusammengefasst. Jede GOP stellt eine abgeschlossene Einheit dar. Wenn ein Frame aus solch einer Gruppe extrahiert werden soll, sind meist aufwendige Berechnungen nötig. Deshalb wird das MPEG-Format auch selten in seiner allgemeinsten Form in Videoschnittprogrammen eingesetzt.

I-Frames oder auch Intracoded Frames sind einfache JPEG-komprimierte Frames. Hierbei wird auch die YUV-Kodierung (s.o.) der Farbinformationen verwendet. Da das menschliche Auge auf Farben weniger empfindlich als auf Helligkeit ist, werden jeweils 4 der 8x8 Pixel großen Kompressionsblöcke der DCT im Y-Kanal einem 8x8 Pixel großen Block des U-Kanals und einem 8x8 Pixel Block des V-Kanals zugeordnet. U- und V-Kanal werden also in einer geringeren Auflösung abgespeichert. In Abbildung 41 ist ein Ausschnitt aus einem MPEG-kodierten Frame schematisch dargestellt. Man erkennt, wie die Pixel der Farbkanäle innerhalb der Helligkeitskanäle verteilt sind.

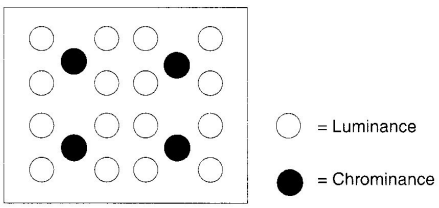


Abbildung 41. Verteilung der Farb- und Helligkeitspixel in einem MPEG-Frame

In P-Frames, das P steht für *predicted* ((engl.) = vorhergesagt), wird die zeitliche Redundanz eines Films ausgenutzt. In MPEG-1 wird das P-Frame in so genannte 16x16 Pixel große Makroblöcke eingeteilt. Ein Suchalgorithmus sucht in einem vorangegangenen I- oder P-Frame nach einem ähnlichen Makroblock, indem alle möglichen Makroblöcke im vorangegangenen Frame (d.h. auch um nur wenige Pixel verschobene) in einer Umgebung des zu kodierenden Makroblocks mit ihm verglichen werden.

- Wird der gleiche Makroblock an der selben Stelle im vorherigen I- oder P-Frame gefunden werden in diesem P-Frame für diesen Makroblock keine Daten gespeichert.
- Wird der gleiche oder ein ähnlicher Makroblock an einer anderen Stelle gefunden, wird in dem aktuellen Makroblock eine Referenz in Form eines Verschiebungsvektors auf den vorangegangenen Makroblock gespeichert. Falls sich der Makroblock leicht verändert hat, werden

die Unterschiede berechnet und als JPEG-ähnliches Bild gespeichert. Da die Unterschiede gering bzw. regelmäßig sind, kann die JPEG-Kompression gute Ergebnisse erzielen. Hierbei hilft vor allen Dingen die Entropiekodierung, da viele Nullen entstehen.

- Hat sich ein Makroblock zu sehr verändert, wird er wie im I-Frame als JPEG-Bild kodiert.

In MPEG-4 können mittels einer alpha Maske (s.u.) beliebig geformte Objekte definiert werden. Somit kann ein Objekt (z.B. eine Person) durch einen Bewegungsvektor mit einem sehr geringen Differenzbild (der Hintergrund ist weggeschnitten) sehr effizient kodiert werden. Voraussetzung ist natürlich, dass der Encoder die Objekte erkennt oder dass sie von vorn herein bekannt sind. Dies ist beispielsweise bei animierten Filmen der Fall.

Abbildung 42. Dieses Bild veranschaulicht das Prinzip der P-Frame Kompression. Oben links ist der zu kodierende Frame, oben rechts die Bewegungsvektoren, unten rechts das Differenzbild und unten links der Unterschied zum vorherigen Frame zu sehen.

B-Frames erreichen die größten Kompressionsraten. Es werden wie bei den P-Frames ähnliche Makroblöcke in einem vorangegangenen und einem nachfolgenden I- oder P-Frame benutzt. Der Mittelwert sowohl der Position, als auch der Farbwerte ist die Grundlage des Makroblocks im B-Frame. Zusätzlich können wie im P-Frame Differenzbilder gespeichert werden. Da die aus bereits gespeicherten Daten berechnete Grundlage besser als die im P-Frame durch Bewegungsvektoren berechnete ist, sind die Differenzbilder wesentlich einfacher strukturiert und werden deshalb auch besser komprimiert.

Der MPEG-Standard gibt keine Bedingungen vor, wie viele I-, P- und B-Frames wann kommen. Einzige Einschränkung bei MPEG-2 ist, dass B Frames weder als Bezugsframe benutzt werden dürfen, noch dass sie am Anfang einer GOP stehen dürfen. Durch die erstere Bedingung können B-Frames mit mehr Verlust komprimiert werden, ohne dass sich der Fehler weiter fortpflanzt und so verstärkt,

letztere Bedingung ergibt sich direkt aus der Definition der B-Frames. Somit kann der Encoder die Filmkodierung optimieren, indem er verschiedene Reihenfolgen von I-, P- und B-Frames benutzt.

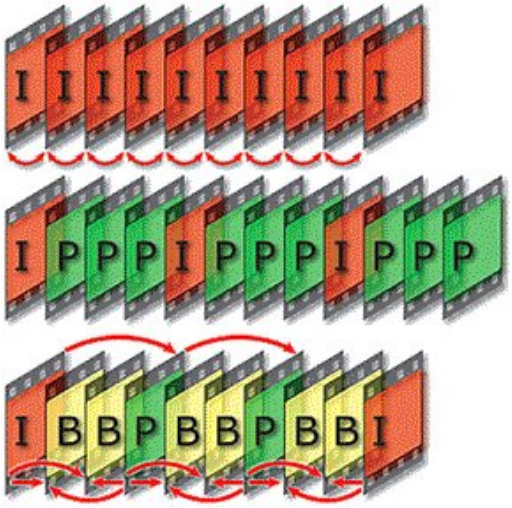


Abbildung 43. Die ursprünglichen Einzelbilder eines Films werden zunächst in I-Frames (JPEG-Bilder) kodiert und danach in eine IBBPBBPBBI Sequenz komprimiert. Die Pfeile deuten jeweils Referenzen auf andere Frames an.

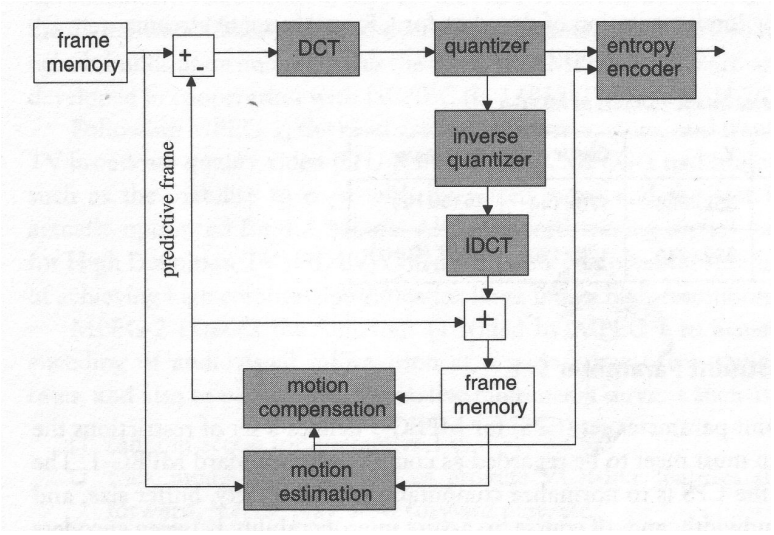


Abbildung 44. Schematischer Aufbau eines MPEG-1-Encoders

4.4. Verbesserungen in H.264 / MPEG-4 AVC

Das neuste Kompressionsverfahren ist keine komplett neue Erfindung, es werden lediglich einige zum Teil wesentliche Komponenten des MPEG-4 Part 2 ausgetauscht oder verbessert. Die DCT wird durch eine neue Transformation ersetzt, die auf 4x4 großen Quadraten operiert und einfacher in Hardware zu integrieren ist. Anstatt der Huffman-Entropiekodierung wird die arithmetische Kodierung (siehe oben) verwendet, die in Bezug auf die Entropie optimal ist und sie nicht nur wie die Huffman-Kodierung bis auf ein Bit erreicht. Die Huffman-Kodierung wird besonders ineffizient, wenn wir nur zwei Symbole zu kodieren haben, bei dem das eine eine Antreffwahrscheinlichkeit größer 50%

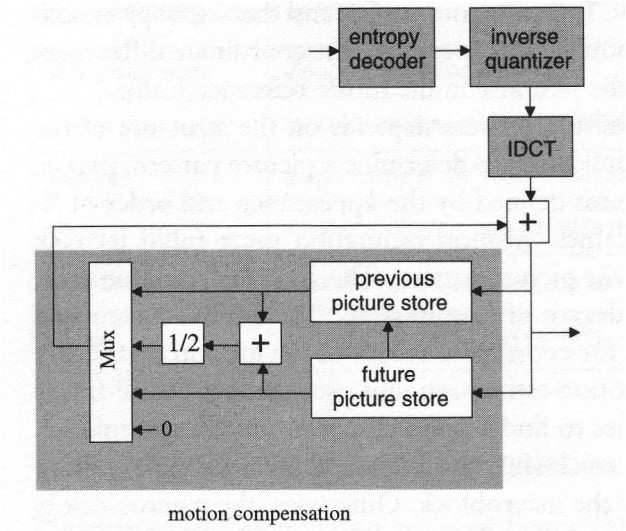


Abbildung 45. Schematischer Aufbau eines MPEG-1-Decoders

hat (siehe oben). Dies ist bei Bildern sehr häufig der Fall, da mit DCT, Quantisierung, Differenzbildern etc. möglichst viele Nullen erzielt werden sollen. Des Weiteren können die Bezugspunkte von B- und P-Frames flexibler gestaltet werden. In einem Framebuffer werden jetzt mehrere vorangegangene Frames gespeichert. Diese können in anderen Frames als Bezugspunkte benutzt werden. Zudem ist die Grenze zwischen I-, P- und B-Frames aufgehoben. Durch so genannte *Slices* können in einem Frame einige Makroblöcke mit dem P-Frame-Prinzip kodiert werden, andere mit dem B-Frame-Prinzip und wieder andere mit dem I-Frame-Prinzip. Die I-Frames können nun auch intern eine Art Motion-Vektor Kompression benutzen, das heißt auch in einem zweidimensionalen Bild sind Referenzen auf einen anderen Teil des selben zweidimensionalen Bildes zu finden. Hierbei können die einzelnen Referenzen auch Transformationen, wie Rotationen oder Skalierungen benutzen. Dieses Kompressionsverfahren heißt *Wavelet*-Kompression und fürt zu einer besseren Skalierbarkeit und Kompression der Bilder. Die Makroblöcke können jetzt 4x4, 4x8, 8x8, 16x8 oder 16x16 groß sein. Des Weiteren haben die Bewegungsvektoren eine Auflösung von $\frac{1}{4}$ Pixel statt $\frac{1}{2}$ Pixel oder weniger in vorangegangenen Standards. Um Blockbildung zu verhindern, werden verschiedene Filter eingesetzt. In H.264 sind diese standardisiert und werden auch bei der Encodierung verwendet. So werden P- und I-Frames aufgrund der tatsächlich angezeigten Bilder vorhergesagt bzw. interpoliert und nicht aufgrund der Rohdaten, die der Decoder bekommt und auf die er im Allgemeinen für den Encoder unbekannte Filter anwendet. Ein weiterer Vorteil der standardisierten Filter ist, dass je nach Objekt (z.B. je nach Szene, aber auch für ein Gesicht im Vordergrund und einen Himmel im Hintergrund (genauere Beschreibung unten)) unterschiedliche Filter angewendet werden können, die mehr oder weniger stark glätten.

Der Nachteil bei immer besseren Filmkodierungsverfahren ist der immer größer werdende Rechenaufwand sowohl bei der Encodierung als auch bei der Decodierung. Somit ist der Einsatz, z.B. in Handys und DVD-Playern nicht immer in vollem Umfang möglich und sinnvoll.

4.5. Der Dschungel der Filmformate

Bevor ich in weitere Einzelheiten gehe möchte ich noch kurz auf gängige Akronyme bei der Filmkodierung eingehen. MPEG-1, -2, -4 sind Standards, die von der *Motion Picture Expert Group* zum

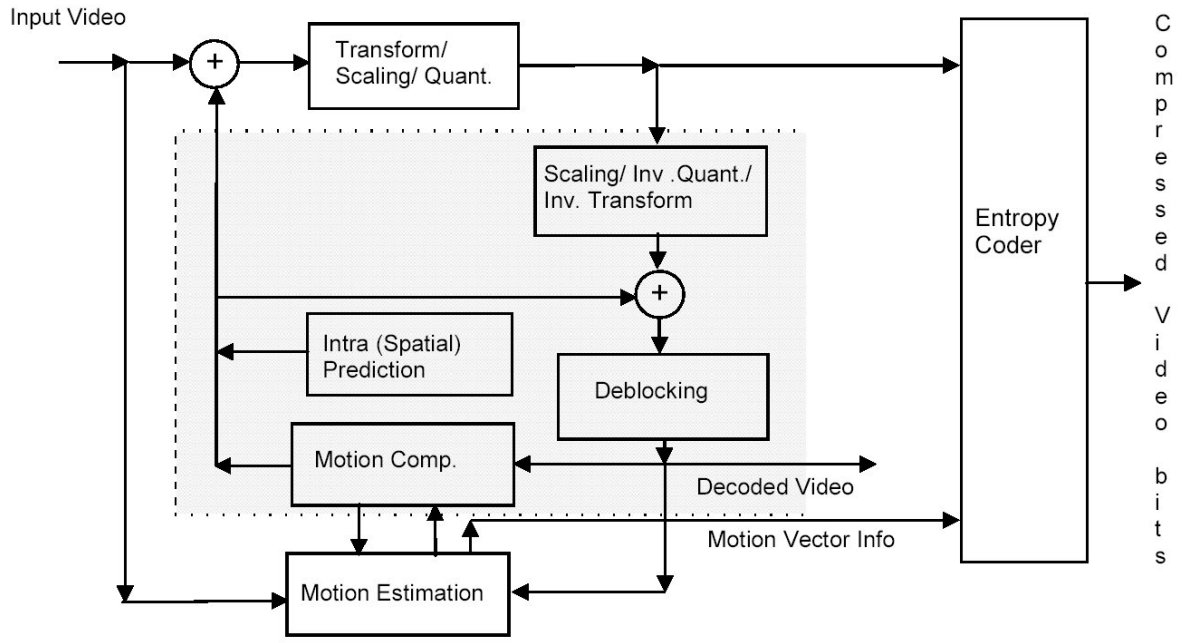


Abbildung 46. Schematischer Aufbau eines H.264 Encoders

Kodieren von Filmen definiert wurden. Von MPEG-1 auf MPEG-2 hat sich die Auflösung des Bildmaterials vergrößert und es wurden zusätzliche Neuerungen wie ein Filmrechtesystem und ein Layersystem eingeführt. Das Layersystem besteht aus einem Baselayer und einem oder mehreren Enhancementlayern. Im Baselayer wird z.B. ein niedrig aufgelöster Film übertragen. Im Enhancementlayer werden die nach der Hochskalierung auf größere Auflösungen nötigen Korrekturen übertragen. So ist es möglich, einen Film gleichzeitig in TV- und in HDTV-Qualität zu übertragen. Dieses Prinzip nennt man Skalierung. Gleiches ist auch in zeitlicher Dimension möglich. Der Baselayer enthält beispielsweise jedes vierte Frame, der Enhancementlayer die restlichen. Eine weitere Anwendung für das Layerprinzip ist das Filmstreaming im Internet. Wenn die Verbindung schlecht ist, wird nur der Baselayer übertragen. MPEG-2 wird unter anderem auf DVDs, bei der senderinternen Übertragung von Filmen und Berichten via Satellit und im PayTV Bereich eingesetzt. MPEG-3 war eine Entwicklunsversion und ist heute in MPEG-2 integriert. MPEG-4 hat ähnliche Ideen wie MPEG-1 und -2 zur Filmkodierung. Zusätzlich werden Objekte eingeführt, die durch Pixel-Masken markiert werden. Auch normale Framefolgen werden jetzt als Objekte bezeichnet. Objekte können beliebig geformt sein und Inhalt in unterschiedlichen Formaten und/oder in unterschiedlichen Qualitätsstufen enthalten. So gibt es in MPEG-4 Part 10 die Möglichkeit, eine Animation mit einem normalen Video zu kombinieren. Ein anschauliches Beispiel ist hierzu der Wetterbericht im Fernsehen. Die animierte Wetterkarte ist ein Objekt, der Sprecher ein anderes. Große Verwirrung kann der MPEG-4 Standard bieten, der nicht ein Format, sondern gleich mehrere definiert, die durch so genannte Profile weiter unterteilt sind. Andere Formate wie DivX, Xvid oder H.264 gehen auf MPEG-Formate zurück.

4.6. Objektorientiertheit in MPEG-4

Ein Objekt kann in MPEG-4 eine beliebige Form haben und wird durch eine Pixelmaske markiert (vgl. Abbildung 47). Zudem können Objekte in MPEG-4 auch durch eine affine Abbildung transformiert werden. Somit kann bei Kamerazooms oder Kamerafahrten auch eine Kompression erreicht

werden. Der Objektbegriff in MPEG-4 ist aber noch viel weiter gefasst worden. Objekte können auch Tonspuren, ASCII Text, der mit Text-to-Speach Konversation vorgelesen wird, Untertitel und Animationen sein. Jedes Objekt kann dann mit seinem optimalen Kompressionsverfahren komprimiert werden. Der wesentliche Grund für die bessere Komprimierung von DivX ist die variable Bitrate. Diese wird mit Hilfe von Objekten realisiert. Hierzu wird eine bestimmte Szene (mehrere aufeinanderfolgende Frames) oder aber nur ein bestimmter Bildausschnitt in einer Szene als Objekt aufgefasst und mit unterschiedlicher Qualität kodiert. Hierzu wird typischerweise die Quantisierungsmatrix angepasst, z.B. indem sie mit einem zusätzlichen Faktor multipliziert wird. Welche Qualität nötig ist, wird vom Encoder anhand von verschiedenen Kriterien entschieden. In schnellen Szenen (wenige erkannte Makroblöcke, große Bewegungsvektoren, starke statistische Schwankung von Bildparametern wie der Gesamthelligkeit, des Rotanteils o.ä.) wird die Qualität der Darstellung, insbesondere in den hohen Frequenzen vermindert. Diese Idee beruht auf der Erkenntnis, dass das menschliche Auge bei schnell bewegten Objekten feine Details nicht mehr so gut wahrnehmen kann. Beispielsweise kann man das Nummernschild eines schnell vorbeifahrenden Autos nicht erkennen, obwohl wir es erkennen könnten wenn es stehen würde. Ein anderes Verfahren zur Erkennung von weniger wichtigen Bildausschnitten ist das Lumi-Masking. Das menschliche Auge nimmt in sehr hellen oder sehr dunklen Bildausschnitten feine Strukturen weniger stark wahr. Um ein Objekt mit großer Helligkeit zu finden, das man mit schlechterer Qualität kodieren kann, kann man z.B. den Y-Kanal analysieren.

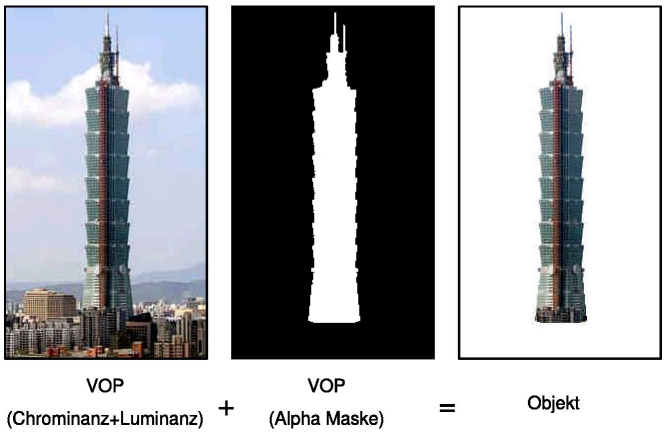


Abbildung 47. Ein Objekt wird in MPEG-4 mittels einer Maske markiert.

4.7. Weitere Features in MPEG-4

4.7.1. Sprites

Es wird ein konstantes Hintergrundbild, ein so genannter *Sprite*, separat und einmal übertragen. Die sich davor bewegenden Objekte zusammen mit ihrer Maske und gewissen Transformationen für den Hintergrund (Skalierung, Ausschnitt etc.) sind die einzigen Informationen, die danach noch übertragen werden müssen. Es gibt auch Methoden, den Sprite sukzessive zu übertragen.

4.7.2. Meshgitter

Wie bei Computerspielen können Objekte durch ein Meshgitter (siehe Abbildung 48) beschrieben werden. Ein solches Gitter wird auch bei *Face Animation* verwendet (siehe dazu Abbildung 49).

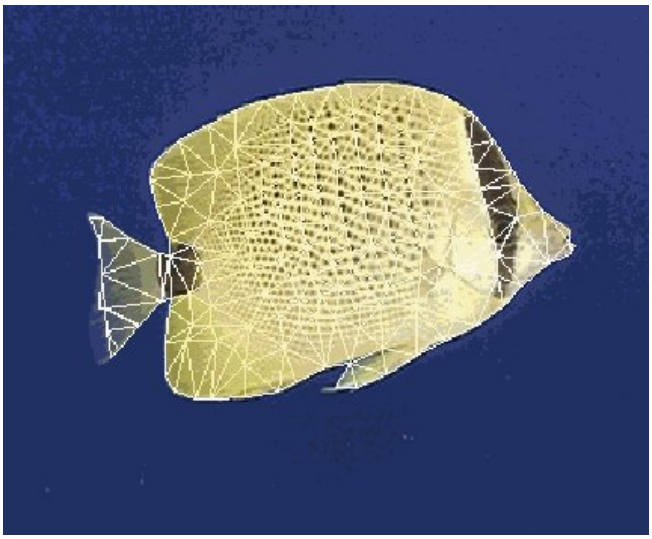


Abbildung 48. Mesh-Gitter

4.7.3. Face Animation

Anstatt ein Pixelbild eines Gesichts zu übertragen, wird einmalig ein Gitterbild des Gesichts mit einem neutralen Gesichtsausdruck und der dazugehörigen Textur gesendet. In der darauffolgenden Übertragung werden nur noch Transformationen einzelner Gitterpunkte übertragen. Dieses Verfahren wurde speziell für Videokonferenzen entwickelt.

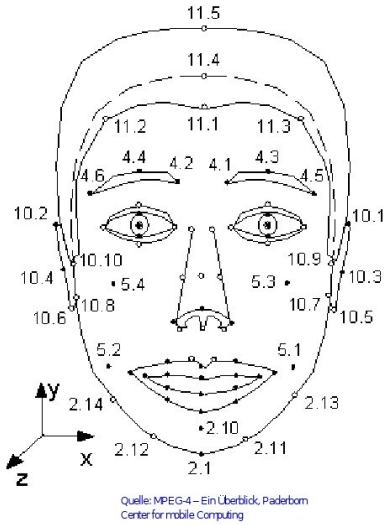


Abbildung 49. Beispiel eines Gesichtsgitters

4.7.4. Scene Description Language

Wenn mehrere Objekte übertragen werden, kann durch die *Scene Description Language* die Position der Objekte im Raum beschrieben werden, sowie die des Betrachters. Hierdurch ergibt sich auch die Möglichkeit, als Betrachter interaktiv in die Szene einzugreifen.

4.8. Entwicklung der Filmkompression

Die wesentlichen Entwicklungsstufen der Filmkompressionsverfahren sind:

- MPEG-2: I-, B- und P-Frames, *Motion Prediction* mit Bewegungsvektoren mit 1 Pixel Auflösung und Makroblöcke die 8x16 oder 16x16 Pixel groß sind. Dieser Standard wird vor allem auf DVDs eingesetzt. Zudem wird das Layer Konzept eingeführt und die JPEG-Kompression ist für Halbbilder durch verschiedene Zickzack-Muster optimiert.
- MPEG-4 Part 2: Hier werden Objekte eingeführt. Nun ist es möglich, mit variabler Bitrate Filme zu speichern. Ein typischer Vertreter ist beispielsweise DivX. Eigentlich schreibt der Standard auch eine *Globel Motion Control* vor (ein Bewegungsvektor für das ganze Frame), diese wird jedoch nicht immer implementiert. *Sprites* sollten auch schon in diesem Standard möglich sein, werden jedoch selten implementiert und benutzt, da die Hintergrunderkennug schwer beim Kodieren ist, und die Dekodierung aufwendig und speziell in Hardware einiges an Anstrengungen verlangt.
- MPEG-4 Part 10: Es wird die arithmetische Kodierung als Entropiekodierung eingeführt; Filter gegen die Verpixelung, die auch im Encoder verwendet werden, werden eingeführt; bei der Motion Prediction wird mit $\frac{1}{4}$ genauen Bewegungsvektoren gearbeitet, des Weiteren können die Makroblöcke 16x16, 16x8, 8x8, 8x4 und 4x4 Pixel groß sein. Die DCT wird durch eine schnellere Transformation ersetzt. Sprites können als Wavelet-Bilder abgespeichert werden.
- MPEG-4 Part 10 Profile Visual: Hierunter kann man alle zusätzlichen Features von MPEG wie Mesh-Gitter, Face-Animation, Scene Description Language zusammenfassen. Ich werde diese Verfahren weiter unten kurz erläutern.

Avi, VOB oder MP4 sind Containerformate. In ihnen wird z.B. ein MPEG-kodierter Film zusammen mit einer mp3-kodierten Tonspur gespeichert und zeitlich synchronisiert.

Quicktime ist zum einen ein eigenes Containerformat, zum anderen definiert es Formate, die verwendet werden können, und dementsprechend von einem Quicktime-Player abgespielt werden. Hierzu gehören beispielsweise *Flash*, mp3 und MPEG-4 Part 2.

Der neuste Codec heißt MPEG-4/AVC, der auch H.264 genannt wird.

Der neuste Standard, der entwickelt wird, ist MPEG-7. Er ist ähnlich zu *Quicktime* ein Sammelsurium von Formaten, geht aber darüber hinaus. In ihm werden auch Interfaces definiert, durch die ganz neue Formate benutzt werden können. So kann z.B. angegeben, wo man den entsprechenden Decoder herunterladen kann. MPEG-5 und -6 sind Entwicklungsversionen.

4.9. Filmbeschreibungssprachen als Alternative?

Mit den verschiedenen Objekttypen von MPEG-4 kommt man immer näher an Filmbeschreibungssprachen. Statt einen Film als Menge von Pixeln aufzufassen, kann man ebenso die Objekte (wie Kugeln, Lichter), die in einer Szene sind, in kurzen Befehlen beschreiben und erst beim Abspielen zu einem Bild zusammenrechnen. So geschieht es z.B. bei Computerspielen. Das Problem ist hierbei die große benötigte Rechenleistung. Selbst mit heutigen Computern und Grafikkarten erkennt man

immer einen Unterschied zwischen einer Filmszene und einer PC-Spiel-Szene. Selbst professionelle Trickfilme, die auf großen Rechnerfarmen gerendert werden, erscheinen nicht real. Eine typische Filmbeschreibungssprache ist beispielsweise Flash.

4.10. Für Filmpräparation verwendete Programme

- DVDRipper nicht ganz legal, ist aber nötig um das Rohmaterial die VOB von der DVD zu extrahieren.
- PVRDemuxer kann den MPEG-2 Stream aus dem VOB-File extrahieren.
- MPEG-2 Schneideprogramm um die entsprecheden Szenen herauszuschneiden.
- SUPER inklusive x264 Erweiterung für das H.264 Format von eRights basiert auf ffmpeg zum umwandeln von MPEG in die verschiedenen Formate MPlayer mit GUI basiert ebenfalls auf ffmpeg kann gut konfiguriert werden um auch einzelne Frames anzusteuern.

ffmpeg und MPlayer sind freie Software unter der GPL und für Windows und Linux verfügbar.

5. Fazit

Die Vielzahl der auf dem Markt verfügbaren Videokompressionsverfahren haben alle das gleiche Ziel: Sie wollen Redundanzen nutzen und geschickt kodieren, damit sie einen Film komprimiert speichern können. Hierbei werden auch Schwächen des menschlichen Auges genutzt und so irrelevante Informationen herausgefiltert und nicht abgespeichert. Als gute Methode hat sich hierbei eine Kombination aus einer Transformation (z.B. Diskrete Kosinus Transformation (DCT)) inklusive Quantisierung, eine Entropiekodierung (z.B. Huffman) und eine auf Makroblöcken und Bewegungsvektoren basierende Kompression in der zeitlichen Dimension herausgestellt. In moderneren Standards werden hierfür Objekte definiert, die auf verschiedene Arten abgespeichert werden. Neben der Wahl von verschiedenen Qualitäten können auch komplett andere Formate (z.B. für Animationen) gewählt werden.

Schlussbemerkung: In der Praxis kann man natürlich auch eine geringere Auflösung oder eine geringere Framerate zur Kodierung benutzen. Dies spart zusätzlich Speicherplatz.

Literatur

[1] Effelsberg, Wolfgang und Ralf Steinmetz, *Video Compression Techniquis*, dpunkt.verlag, Heidelberg, 1998

[2] Sikora, Thomas, *Trends and Perspectives in Image and Video Coding*, Januar 2005

[3] Gary J. Sullivan, Pankaj Topiwala, Ajay Luthra, *The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions*, August 2004

[4] Guido H. Bruck, *Projektseminar Bildverarbeitung Image Processing*, Uni Duisburg, Mai 2003

[5] The Comunity, *www.wikipedia.de*, 2006

[6] Vasquez, Dizan & Thierry Fraichard, *Motion Prediction for Moving Objects: a Statistical Approach, http://www.inrialpes.fr/sharp*

Alle Paper und Folien sind im Internet frei zugänglich. Ich habe sie über *google* oder *CiteSeer* gefunden.

Teil IV

Subdivision Surfaces

Sebastian Reiter[5]

Abstract. *Glatte Oberflächen beliebiger Topologie können auf viele verschiedene Arten beschrieben werden. Eine Möglichkeit sind die so genannten* Subdivision Surfaces*. Als Eingabe dient eine stetige, stückweise lineare Oberfläche beliebiger Topologie. Durch wiederholte Verfeinerung erhalten wir daraus eine Folge von stetigen, stückweise linearen Oberflächen, die gegen eine glatte Oberfläche streben - die* Subdivisoin Surface*. Im Folgenden werden wir einen Überblick über die Theorie hinter* Subdivision Surfaces *geben, eine grobe Einteilung verschiedener Schemata bereitstellen sowie einige Verfahren genauer untersuchen.*

1. Einleitung

1.1. Was sind Subdivision Surfaces?

Subdivision Surfaces (Unterteilungsflächen) dienen der Beschreibung von glatten Oberflächen beliebiger Topologie. Mittels eines so genannten *Kontroll-Meshes* lässt sich die Topologie sowie die ungefähre Form der Subdivision-Surface vorgeben. Indem man von dem Kontroll-Mesh ausgehend wiederholt verfeinert und nach jeder Verfeinerung die Punkte des neuen Gitters nach gewissen Regeln verschiebt, erhält man - bei passender Wahl des Regelwerks - eine glatte, den Kontroll-Mesh approximierende Oberfläche.

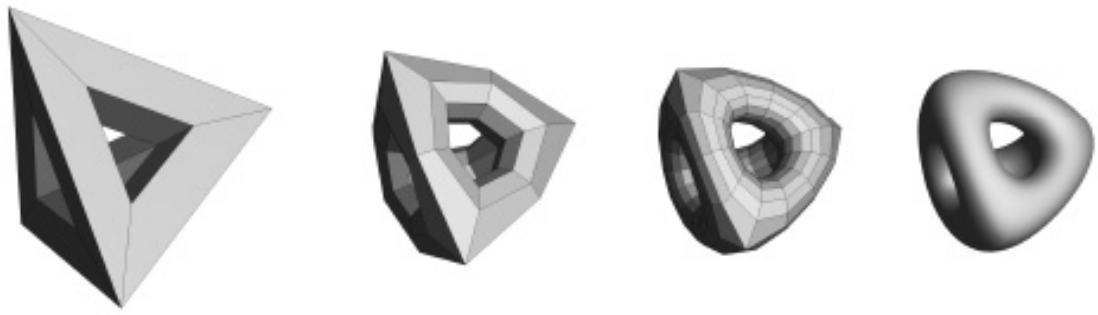


Abbildung 50. Beispiel für Subdivision

Das erste Mal beschrieben wurden Subdivision Surfaces 1978 in Arbeiten von Doo und Sabin sowie Catmull und Clark. Erst 1995 gelang es aber grundlegende Fragen über das Verhalten von Subdivision Surfaces in der Umgebung außerordentlicher Knoten zu beantworten (Reif). Seither wurden viele neue Schemata entwickelt sowie die Glattheit der meisten Verfahren untersucht.

[5]IWR, Universiät Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, sreiter@ix.urz.uni-heidelberg.de

1.2. Begriffe

Zunächst sollen einige Begriffe erläutert werden:

Mesh: Ein Mesh beschreibt eine stückweise lineare Oberfläche. Er besteht aus *Knoten* (Vertices), *Kanten* (Edges) und *Flächenstücken* (Faces). Im Folgenden wird davon ausgegangen, dass jede Kante eines Meshes höchstens zwei, mindestens aber ein benachbartes Flächenstück hat.

Glattheit: Eine Oberfläche $O \subset \mathbb{R}^3$ wird im Folgenden als glatt bezeichnet, wenn zu jedem Punkt $p \in O$ eine offene Umgebung $U \subset \mathbb{R}^3$ und eine offene Nullumgebung $V \subset \mathbb{R}^2$ existieren, so dass eine stetig differenzierbare Abbildung $\varphi : \mathbb{R}^2 \to \mathbb{R}^3$ existiert mit $\varphi(V) = U \cap O$.

1.3. Anforderungen an Subdivision Surfaces

Um sinnvoll angewandt werden zu können, sollten Subdivision-Surfaces einigen Anforderungen genügen. Dazu zählen:

- *Effizienz des Regelwerks*: Die Berechnung der neuen Positionen der Knoten nach einem Verfeinerungsschritt sollte wenige Operationen benötigen.
- *Kompakter Träger*: Die Umgebung, in der ein Knoten die Form der resultierenden Oberfläche beeinflusst, sollte möglichst klein, in jedem Fall endlich sein.
- *Lokale Definition*: Die Regeln für die Positionierung neuer Knoten sollte nicht auf weit entfernten Knoten (graphentheoretisch) beruhen.
- *Affine Invarianz*: Sollte der Kontrollmesh *M* einer affinen Transformation (z.B. Translation, Skalierung, Rotation) unterzogen werden, so sollte sich auch die aus dem ursprünglichen Mesh resultierende Subdivision Surface durch die selbe Transformation in die aus dem transformierten Mesh resultierende Subdivision Surface transformieren lassen.
- *Einfachheit*: Das Regelwerk sollte möglichst klein sein.
- *Stetigkeit*: Es wäre wünschenswert, Aussagen über die Stetigkeit / Glattheit der resultierenden Subdivision Surface treffen zu können.

2. Grundlagen

Wir betrachten zunächst eine Folge von *Kontrollpunkten* $p_i \in \mathbb{R}^2$. Unser Ziel ist hierbei, diese Kontrollpunkte durch eine Funktion $f : \mathbb{R} \to \mathbb{R}^2$ zu interpolieren (d.h. die von der Funktion beschriebene Kurve läuft durch die Punkte) oder zu approximieren (die Funktion nähert die Folge der Kontrollpunkte). Um dies zu bewerkstelligen, bedienen wir uns sogenannter *Splines*. Polynomiale Splines sind Funktionen, die stückweise aus *Polynomen* zusammengesetzt werden und je nach Bauart die oben geforderten Eigenschaften erfüllen.

2.1. B-Splines

Wir betrachten im Folgenden speziell *B-Splines*. Diese haben einige Eigenschaften, die man auch bei Subdivision Surfaces beobachten kann.

2.1.1. Darstellung von B-Splines

Sei $f(t) = (x(t), y(t))$. Dann kann man die Funktionen x und y wie folgt darstellen:

$$x(t) := \sum x_i B_i(t)$$

$$y(t) := \sum y_i B_i(t).$$

Mit $p_i = (x_i, y_i)$. Die *Basisfunktionen* B_i werden dabei so gewählt, dass sie lokalen Träger haben. Dadurch beschränkt sich der Einfluß, den ein Kontrollpunkt p_i auf die Kurve hat, auf eine Umgebung von p_i. Eine weitere wichtige Eigenschaft ist die Stetigkeit von f. Diese ist gegeben, wenn alle B_i stetig sind. Um einen weichen Kurvenverlauf zu erhalten, sollten die B_i hinreichend glatt sein.

2.1.2. Konstruktion der Basisfunktionen

Um die Basisfunktionen B_i der B-Splines zu konstruieren, benutzen wir die Technik der *Faltung*. Die Faltung zweier Funktionen f und g lässt sich folgendermaßen darstellen:

$$(f * g)(t) := \int f(s)g(t-s)ds.$$

Sei nun $G_0 : \mathbb{R} \to \mathbb{R}$ gegeben durch

$$G_0(t) \quad := \quad \begin{cases} 1 & \text{für } 0 \leq t < 1 \\ 0 & \text{sonst} \end{cases}$$

Dann erhalten wir G_1, indem wir G_0 mit sich selbst falten:

$$G_1(t) := \int G_0(t)G_0(s-t)ds$$

War G_0 noch unstetig an den Stellen 0 und 1, so ist G_1 stetig. G_1 ist eine so genannte Hutfunktion. Faltet man nun G_1 mit G_0, so erhält man eine ein mal stetig differenzierbare Funktion G_2. Allgemein halten wir fest:

$$G_l(t) := \int G_{l-1}(t)G_0(s-t)ds.$$

Eine wichtige Eigenschaft dieser Konstruktionsmethode ist die folgende: Wenn $f(t) \in C^k$ dann ist $(G_0 * f)(t) \in C^{k+1}$. Für die Funktion G_n gilt also: $G_n \in C^{n-1}$.

Die Funktionen B_i erhalten wir nun aus einem der G_n folgendermaßen:

$$B_i(t) := G_n(t-i) \quad \text{für ein } n \in \mathbb{N}.$$

2.1.3. Verfeinerbarkeit von B-Splines

Eine wichtige Eigenschaft der B-Splines ist ihre *Verfeinerbarkeit.* Diese Eigenschaft ist es, die B-Splines und Subdivision-Surfaces eng verbindet.

Unter der *Verfeinerbarkeit* von B-Splines versteht man, dass man neue Kontrollpunkte so einfügen kann, dass sich die durch den B-Spline beschriebene Kurve nicht ändert. Die oben konstruierten Basisfunktionen erfüllen also die *Verfeinerungs-Gleichung*

$$G_l(t) = \frac{1}{2^l} \sum_{k=0}^{l+1} \begin{pmatrix} l+1 \\ k \end{pmatrix} G_l(2t-k).$$

Wir können B-Spline Basen also als Summe über gestauchte und verschobene Kopien ihrer selbst schreiben.

Wir betrachten einen B-Spline nun wieder in der Darstellung

$$f(t) := \sum_i p_i B_i(t).$$

Mit dem Vektor

$$p = \begin{pmatrix} \vdots \\ p_{-1} \\ p_0 \\ p_1 \\ \vdots \end{pmatrix}$$

und dem Vektor

$$B(t) = [\ldots, B_0(t+2), B_0(t+1), B_0(t), B_0(t-1), B_0(t-2), \ldots]$$

können wir die Kurve f auch schreiben als

$$f(t) = B(t)p.$$

Motiviert durch die Verfeinerungsgleichung führen wir den Vektor

$$B(2t) = [\ldots, B_0(2t+2), B_0(2t+1), B_0(2t), B_0(2t-1), B_0(2t-2), \ldots]$$

ein und erhalten über die Matrix S den Zusammenhang

$$B(t) = B(2t)S.$$

Es ist zu beachten, dass der Vektor $B(2t)$ mehr Elemente fasst als der Vektor $B(t)$. Die Einträge der Matrix S sind durch die Verfeinerungsgleichung gegeben:

$$S_{2i+k,i} = s_k = \frac{1}{2^l} \begin{pmatrix} l+1 \\ k \end{pmatrix},$$

wobei l den Grad der Basisfunktion bezeichnet. f lässt sich nun schreiben als

$$f(t) = B(t)p = B(2t)Sp.$$

Wie man sieht gehen wir also mit der neuen Basis von den alten Kontrollpunkten p auf die neuen Kontrollpunkte Sp über, verändern aber die beschriebene Kurve nicht.

Wir können diesen Schritt beliebig oft wiederholen:

$$\begin{aligned} f(t) &= B(t)p^0 \\ &= B(2t)p^1 &&= B(2t)Sp^0 \\ &\vdots \\ &= B(2^j t)p^j &&= B(2^j t)S^j p^0 \end{aligned}$$

Für die Beziehung zwischen zwei aufeinanderfolgenden Subdivision-Levels ergibt sich so:

$$p^{j+1} = Sp^j$$

Bei gesonderter Betrachtung der Punkte mit *geradem* Index (diese entsprechen den alten Kontrollpunkten aus p^j) und der Punkte mit *ungeradem* Index (Punkte, die in p^{j+1} durch Verfeinerung neu hinzugekommen sind), erhält man

$$p_{2i}^{j+1} = \sum_l S_{2i,l} p_l^j = \sum_l s_{2(i-l)} p_l^j$$

für die *geraden* und

$$p_{2i+1}^{j+1} = \sum_l S_{2i+1,l} p_l^j = \sum_l s_{2(i-l)+1} p_l^j$$

für die *ungeraden* Knoten.

2.1.4. Subdivision für Spline-Kurven

Wenn wir also den Prozess der Verfeinerung immer weiter wiederholen, erhalten wir eine immer dichter werdende Folge von Kontrollpunkten, die gegen die Spline-Kurve konvergiert. Der Abstand der Kontrollpunkte zur Kurve nimmt dabei um einen konstanten Faktor pro Verfeinerungsschritt ab. Schon nach wenigen Schritten wird es somit schwer, die Kontrollpunkte von der Kurve zu unterscheiden.

2.1.5. Beispiel: Kubische Splines

Bei Kubischen Splines (Grad 3) ergibt sich für die Einträge der Subdivision Matrix

$$s_0 = \frac{1}{8}, \quad s_1 = \frac{4}{8}, \quad s_2 = \frac{6}{8}, \quad s_3 = \frac{4}{8}, \quad s_4 = \frac{1}{8}.$$

Für die *geraden* Knoten ergibt sich so

$$p_{2i}^{j+1} = \frac{1}{8} p_{i-1}^j + \frac{6}{8} p_i^j + \frac{1}{8} p_{i+1}^j.$$

Für die *ungeraden* ergibt sich

$$p_{2i+1}^{j+1} = \frac{1}{2} p_i^j + \frac{1}{2} p_{i+1}^j.$$

Abbildung 51. Ein linearer Spline erscheint bei häufiger Unterteilung als glatte Kurve.

2.2. Mathematische Werkzeuge

Im Folgenden werden wir unsere Theorie erweitern um für den 2-dimensionalen Fall gerüstet zu sein. Desweiteren erlaubt die im Folgenden eingeführte Theorie viele Eigenschaften bezüglich der Subdivision an einfachen Polynomen zu untersuchen.

2.2.1. Generierende Funktion

Wir können eine Folge (a_k) durch eine Funktion $A : \mathbb{R}^n \to \mathbb{R}^n$ mittels folgendem Zusammenhang generieren:

$$A(z) = \sum_k a_k z^k$$

A heißt dann die generierende Funktion der Folge (a_k).

2.2.2. Diskrete Faltung

Die Faltung zweier Folgen (a_k) und (b_k) ist definiert als

$$c_k = (a * b)_k = \sum_n a_{k-n} b_k.$$

Drücken wir die Faltung zweier Folgen über ihre generierenden Funktionen aus vereinfacht sich die Darstellung zu

$$C(z) = A(z)B(z).$$

Eine Eigenschaft die bei dieser Form der Faltung erhalten bleibt ist die der Verfeinerbarkeit. Erfüllen also zwei Funktionen f und g eine Verfeinerungsgleichung

$$f(t) = \sum_k a_k f(2t - k)$$

$$g(t) = \sum_k b_k g(2t - k)$$

so erfüllt auch ihre Faltung $h = f * g$ die Verfeinerungsgleichung:

$$h(t) = \sum_k c_k h(2t - k).$$

Die Koeffizienten c_k ergeben sich dabei aus der Faltung der Koeffizientenfolgen der jeweiligen Verfeinerungsgleichung:

$$c_k = \frac{1}{2} \sum_n a_{k-n} b_n.$$

2.2.3. Generierende Funktion der Verfeinerungsgleichung

Aus Abschnit 2.1.3. lässt sich direkt folgern, dass die Box Funktion $G_0(t)$ die Verfeinerungsgleichung $G_0(t) = G_0(2t) + G_0(2t-1)$ erfüllt. Die generierende Funktion der Koeffizienten dieser Verfeinerungsgleichung lautet $A(z) = (1+z)$.

Aus 2.1.2. wissen wir, dass wir die Basisfunktion eines B-Splines vom Grad l erhalten aus

$$G_l(t) = \bigotimes_{k=0}^{l} G_0(t),$$

wobei $\bigotimes$ die wiederholte Faltungsoperation $*$ darstellt. Es ergibt sich unmittelbar die zugehörige generierende Funktion

$$S(z) = \frac{1}{2^l}(1+z)^{l+1}.$$

Aus dem Binomialsatz folgt:

$$S(z) = \frac{1}{2^l} \sum_{k=0}^{l+1} \begin{pmatrix} l+1 \\ k \end{pmatrix} z^k$$

Die Koeffizienten s_k der Subdivision-Matrix S entsprechen nun gerade den Koeffizienten der verschiedenen Potenzen von z.

3. Subdivision Schemata

3.1. Merkmale und Klassifizierung

Im Folgenden werden Merkmale und Eigenschaften unterschiedlicher Subdivision Schemata aufgeführt mit deren Hilfe wir eine grobe Klassifizierung der verschiedenen Verfahren vornehmen können. Zunächst benötigen wir allerdings weitere Begriffe:

3.1.1. Begriffe

Dreiecks-Mesh: Alle *Faces* des Meshes sind Dreiecke.

Vierecks-Mesh: Alle *Faces* des Meshes sind Vierecke.

Reguläre Knoten: Ein Knoten heißt *regulär*, wenn 6 Kanten (bei Dreiecks-Mesh) bzw 4 Kanten (bei Vierecks-Mesh) von ihm ausgehen.

Face Split: Bei Verfeinerung werden *Faces* in mehrere kleinere *Faces* zerlegt. Alte Knoten bleiben bei Verfeinerung erhalten.

Vertex Split: Bei Verfeinerung werden pro *Face* vier neue Knoten eingefügt (bei Vierecks-Mesh). Neue *Faces* werden erstellt indem die neuen Knoten verbunden werden. Alte Knoten kommen im verfeinerten Mesh nicht mehr vor.

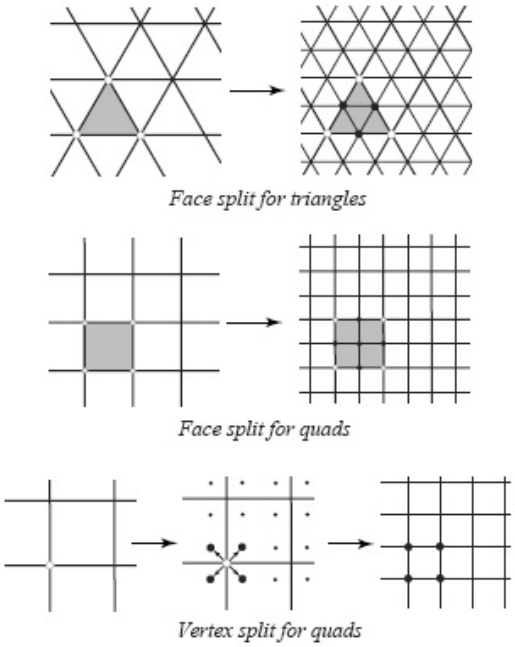


Abbildung 52. ***Face and Vertex Splits*** **bei Dreiecks- und Viereckgittern**

3.1.2. Merkmale

Wir unterscheiden Subdivision Schemata bezüglich folgender Merkmale:

- Art der Verfeinerungsregel (*Face Split* oder *Vertex Split*)
- Typ des zugrunde liegenden Meshes (*Dreiecks-* oder *Vierecks-Mesh*)
- *Approximierende* oder *interpolierende* Schemata
- *Glattheit* der Grenzfläche bei regulären Meshes

3.1.3. Klassifizierung

	Face split	
	Dreiecksgitter	Viereckgitter
Approximierend	*Loop* (C^2)	*Catmull-Clark* (C^2)
Interpolierend	*Mod. Butterfly* (C^1)	*Kobbelt* (C^1)

Vertex split
Doo-Sabin, *Midedge* (C^1)
Biquartic (C^2)

Mit dieser Klassifizierung lassen sich natürlich längst nicht alle Schemata einordnen. Sie dient eher dazu, einen groben Überblick zu bieten.

3.2. Subdivision nach Loop

Beispielhaft werden wir uns in diesem Abschnitt mit dem von Charles Loop 1987 eingeführten *Loop Schema* beschäftigen. Dieses einfache Verfahren arbeitet auf Dreiecks-Meshes. Mittels eines *Face-Splits* wird in einem Verfeinerungsschritt jedes Dreieck des alten Meshes in vier neue unterteilt.

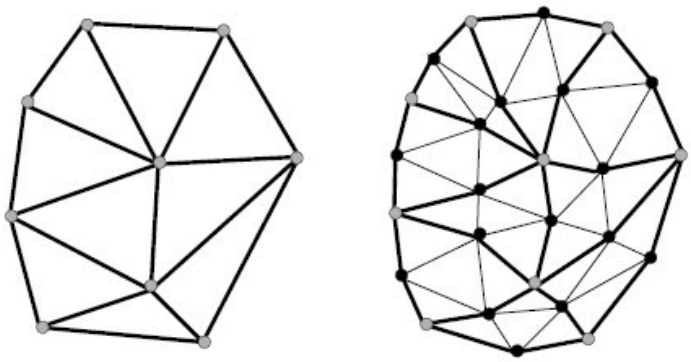


Abbildung 53. Skizze der Verfeinerung beim *Loop-Schema*

Das *Loop Schema* ist ein approximierendes Verfahren, das auf dem *three-directional quartic box spline* basiert.

Die generierende Funktion der zugehörigen Verfeinerungsgleichung lautet:

$$S(z_1, z_2) = \frac{1}{16}(1 + z_1)^2(1 + z_2)^2(1 + z_1 z_2)^2,$$

wobei generierende Funktionen mit zwei Variablen wie folgt definiert sind:

$$A(x, y) = \sum_{n,m=0} a_{n,m} x^n y^m.$$

Es ergeben sich im regulären Fall die in Abbildung 54 dargestellten Gewichte.

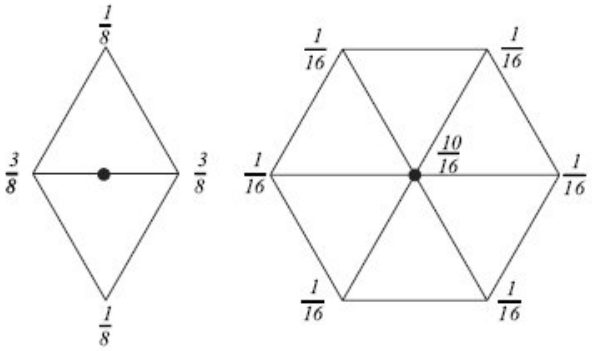


Abbildung 54. Wichtung der Knoten im regulären Fall

Hat im eindimensionalen Fall noch jeder Knoten genau zwei Nachbarn (insofern es sich nicht um einen Randknoten handelt), kann es im zweidimensionalen zu einem Knoten beliebig viele Nachbarn geben. Für die *irregulären* Fälle müssen gesonderte Regeln eingeführt werden.

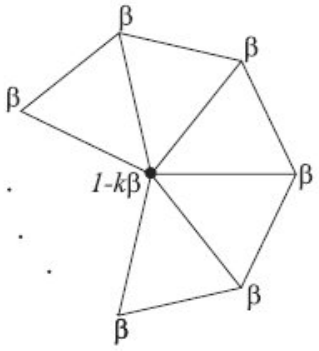


Abbildung 55. Wichtung im Fall eines irregulären Knotens

Loop schlug für die Wahl von β folgende Formel vor: $\beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\Pi}{k}\right)^2\right)$. Abbildung 55 stellt das am Gitter dar. Diese Wahl der Koeffizienten garantiert eine *glatte* Grenzfläche.

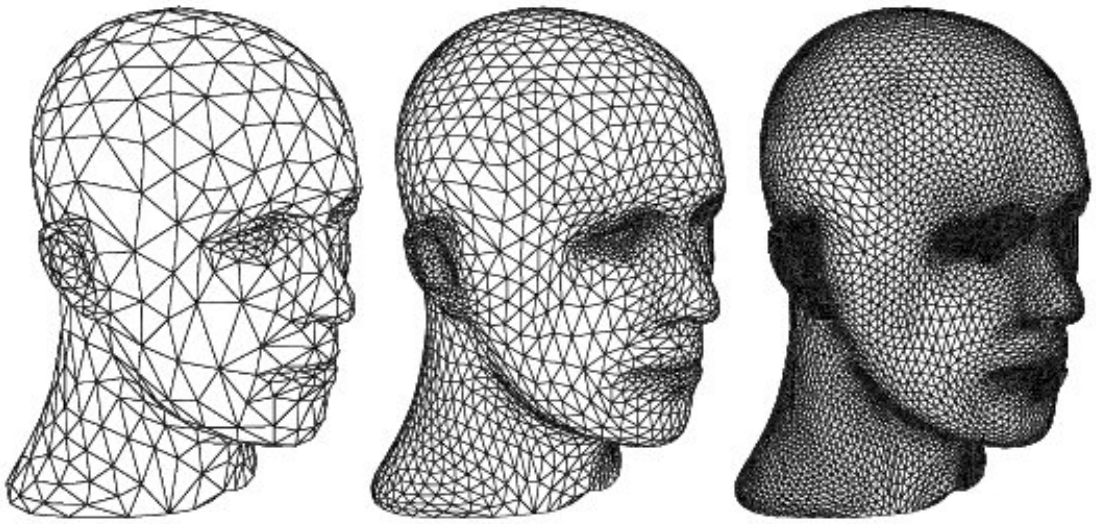


Abbildung 56. Links der Ausgangs-Mesh, rechts zwei Verfeinerungsschritte. Irreguläre Knoten (siehe beispielsweise an der Schläfe) bleiben als solche erhalten.

4. Zusammenfassung und Ausblick

Wir haben neben der zu Grunde liegenden mathematischen Theorie ein Verfahren kennen gelernt, mit dem es uns möglich ist, glatte Oberflächen beliebiger Topologie nicht nur zu beschreiben, sondern mittels einfacher Algorithmen effizient zu approximieren. Um Subdivision Surfaces allgemeiner einsetzen zu können, bedarf es allerdings noch einer Erweiterung des Regelwerks. So haben wir bisher Subdivision ausschließlich für Meshes ohne Rand betrachtet. Auch fehlt uns eine Möglichkeit, scharfe Kanten auf der Oberfläche zu beschreiben. Nähere Informationen zu dieser Problematik, so-

wie eine einfache und effiziente Lösung lassen sich beispielsweise in [2] *'Piecewise Smooth Surface Reconstruction'* finden.

Auch für die Filmindustrie sind Subdivision Surfaces interessant. In [3] *'Subdivision Surfaces in Character Animation'* (auch zu finden in [1]) werden Vorteile von Subdivision Surfaces über traditionelle Oberflächenbeschreibungen (wie z.B. NURBS) bezüglich Animation und Editierbarkeit herausgestellt.

Literatur

[1] SIGGRAPH 2000 Course Notes. Subdivision for Modeling and Animation

[2] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer and W. Stuetzle. Piecewise Smooth Surface Reconstruction.

[3] T. DeRose, M.Kass, T.Truong. Subdivision Surfaces in Character Animation

Teil V

Lösungstechniken für Radiosity

Jens Fangerau[6]

Abstract. *Dieser Artikel befasst sich mit dem globalem Beleuchtungsverfahren* Radiosity.
Anders als bei den lokalen Beleuchtungsverfahren, die jeweils immer nur lokal einen Vertex betrachten, wie z. B. das Gouraud-Shading, wird beim Radiosity-Verfahren der ganze Objektraum berücksichtigt. Dadurch lassen sich realistischere Bilder einer Szene erstellen. Es werden hier nun die Unterschiede zwischen lokalen und globalen Beleuchtungsverfahren erläutert, das Radiosity-Verfahren vorgestellt und Lösungstechniken für die sich ergebende Radiosity-Matrix angegeben.

1. Beleuchtungsmodelle

Um Szenen einer Computergraphik realistisch erscheinen zu lassen, sind LichtmodelLe unverzichtbar. Sie berechnen den Farbwert eines Pixels aufgrund seiner Zugehörigkeit zu einer Fläche nach der Farb- und Oberflächeneigenschaft dieser Fläche sowie der Farbe und Richtung von simulierten Lichtquellen.

1.1. Lokale Beleuchtungsverfahren

Abbildung 57. Szenenbild aus *Doom 3* von *Id Software*

Hier handelt es sich um empirische Modelle. Sie simulieren das Verhalten von direkt gerichtetem Licht auf Oberflächen mit gewissen Materialeigenschaften und betrachten jeweils nur einen einzelnen Punkt einer Fläche. Das ambiente Licht einer Szene wird hier durch einen konstanten Term gegeben (und macht durchschnittlich 30 Prozent in einer Szene aus). Die Verfahren haben den Vorteil, dass sie

[6]IWR, Universiät Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Jens.Fangerau@web.de

relativ schnell sind, wobei aber der Realismus darunter etwas leidet. Beispiele hierfür sind das Flat- und Gouraud-Shading (interpolierte Intensitäten) und das Phong-Shading (interpolierte Normalen).

1.2. Globale Beleuchtungsverfahren

Diese Verfahren beruhen auf komplexeren physikalischen Modellen und simulieren das Verhalten von indirekt reflektiertem und hindurchgelassenem Licht. Dabei betrachten sie im Vergleich zum lokalen Beleuchtungsmodell nicht nur einen lokalen Punkt sondern den ganzen Objektraum, wodurch Schlagschatten, *Color bleeding*, Lichtbrechungen und Spiegelungen erzeugt werden können. Jedoch sind diese Verfahren meist langsamer. Hierzu zählt das im nächsten Abschnitt behandelte Radiosity-Verfahren sowie auch das Raytracing.

Abbildung 58. Steel mill (Solution took 5 hours, 30000 patches, 2000 shots, image generation required 190 hours, each on a VAX8700, siehe [10])

2. Radiosity

2.1. Motivation

Das Radiosity-Verfahren ist ein globales Beleuchtungsverfahren für ideal diffus reflektierende Oberflächen. Es wurde erstmals 1984 von Goral, Torrance, Greenberg und Bataille eingeführt [GORA 84]. Die gesamte von einer Fläche abgegebene Energie heißt Radiosity (Strahlung). Die Idee des Verfahrens beruht auf der Berücksichtigung vom Strahlungsaustausch zwischen Oberflächen und dem Energieerhaltungssatz (Energiesumme in einem abgeschlossenem System ist konstant). Da Licht eine Form von Energie ist, können Sätze der Thermodynamik verwendet werden, um die Radiosity zu berechnen. Zur Vereinfachung der Szene gelten folgende Voraussetzungen:

1. Die Szene wird in *endliche* zusammenhängende Teilflächen (Patches) unterteilt, die so gewählt werden, dass jede Fläche homogen bzgl. ihrer Strahlungsemissions- und Reflexionseigenschaften (konstante Radiosity) ist.

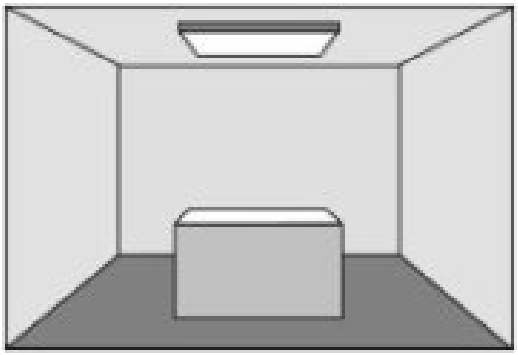


Abbildung 59. Zentralperspektivische Szene

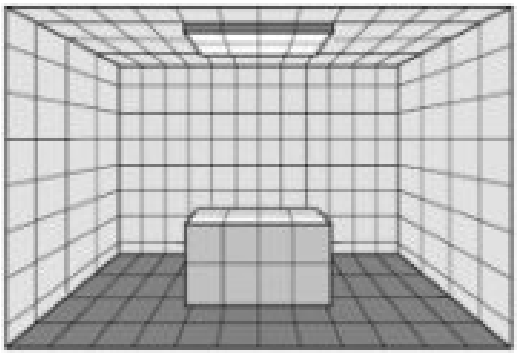


Abbildung 60. Zerlegung der Szene

2. Alle Teilflächen sind *Lambert-Strahler bzw. Reflektoren*, d.h. sie zeigen ideal diffuse Emissions- und Reflexionseigenschaften. Ideal diffus bedeutet, das Licht in alle Richtungen gleichmäßig reflektiert bzw. abgestrahlt wird.

Abbildung 61. Schema einer ideal diffus reflektierenden Fläche

3. Die Szene ist abgeschlossen bzgl. ihrer Strahlungsenergiebilanz, d.h. es wird weder Energie zugeführt noch abgegeben.

Im Gegensatz zu Rendering-Algorithmen berechnet das Radiosity-Verfahren *unabhängig* vom Blickpunkt alle Lichtintensitäten einer Szene. Erst danach werden die Darstellungen gerendert und dabei sichtbare Flächen ermittelt und durch Interpolation schattiert (mittels Flat- oder Gouraud-Shading).

Die Unterteilung der Szene bestimmt den Aufwand des Algorithmus, d.h. je feiner die Unterteilung, desto aufwändiger die Berechnung. Beispielsweise sind analytische Primitive wie z.B. eine Kugel nur schwer zu unterteilen, da eine sehr feine Unterteilung vollzogen werden muss, damit die Struktur der Kugel nicht verloren geht. Deshalb unterteilt man nur dort, wo es erforderlich ist, also z.b. an Schattengrenzen wie sie in der Abbildung 62 ersichtlich sind.

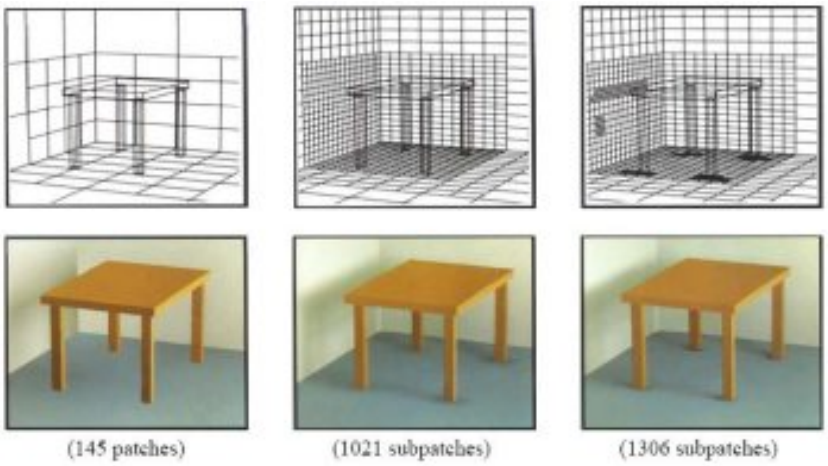


Abbildung 62. Feinere Unterteilung der Szene erhöht den Realismus

2.2. Die Radiosity-Formel

Der Formel für die Radiosity liegt das Strahlungsgleichgewicht in einem abgeschlossenen System zugrunde.

$$B_i A_i = E_i A_i + \rho_i \sum_{j=0}^{n} B_j A_j F_{ji} \qquad (1 \leq i \leq n) \tag{1}$$

B_i : Radiosity der Teilfläche i, also Energie pro Zeit- und Flächeneinheit (Einheit $[W/m^2]$)

A_i : Flächeninhalt von Teilfläche i (Einheit $[m^2]$)

E_i : Emittierte Energie der Teilfläche i ohne Fremdeinwirkung (Einheit $[W/m^2]$)

ρ_i : Reflexionsfaktor der Teilfläche i: gibt an, welcher Teil des einfallenden Lichtes wieder abgestrahlt wird (dimensionslos)

F_{ji} : Formfaktor/Konfigurationsfaktor: gibt an, welcher Anteil der Energiedichte der Fläche j auf die Fläche i übergeht (dimensionslos)

n : Anzahl der Teilflächen in der Szene

Bemerkung 1 *Der Reflexionsfaktor ρ_i und der Emissionsfaktor E_i sind wellenlängenabhängig, jedoch wird der Formfaktor F_{ij} allein von der Geometrie einer Szene bestimmt.*

Aus dem folgenden Reziprozitätsgesetz

$$A_j F_{ji} = A_i F_{ij}$$

folgt dann für (1) :

$$B_i = E_i + \rho_i \sum_{j=0}^{n} B_j F_{ij} \qquad (1 \leq i \leq n) \tag{2}$$

Die Radiosity einer Fläche setzt sich somit aus der eigenen Energie (falls sie eine Lichtquelle ist) und der gewichteten Summe aller auf diese Fläche auftreffenden Energien von anderen Flächen zusammen.

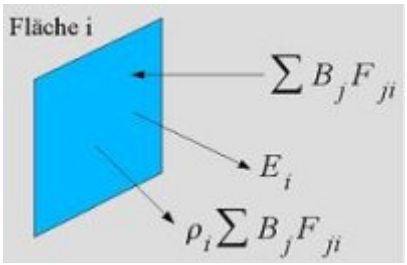


Abbildung 63. Berechnung der Radiosity

2.3. Aufstellung des Gleichungssystems

Gleichung (2) lässt sich umformen zu

$$B_i - \rho_i \sum_{j=0}^{n} B_j F_{ij} = E_i \qquad (1 \leq i \leq n)$$

und lautet in Matrixschreibweise

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ -\rho_n F_{n1} & \dots & \dots & 1-\rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ \vdots \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ \vdots \\ \vdots \\ E_n \end{pmatrix}. \tag{3}$$

Bemerkung 2 *Wenn man in planare Teilflächen unterteilt hat, sind alle* $F_{ii} = 0,\ (i = 1, \dots, n)$ *also alle Diagonalelemente* $= 1$*, da eine planare Fläche sich nicht selbst beleuchten kann.*

Der Formfaktor muss nur einmal berechnet werden, es sei denn, dass sich die Geometrie der Szene ändert. Da der Reflexionsfaktor ρ_i und E_i wellenlängenabhängig sind, muss das Gleichungssystem (3) für jeden Wellenlängenbereich ausgewertet werden, der im Beleuchtungsmodell vorkommt. Jedoch beschränkt man sich hier auf die drei üblichen Primärfarben Rot, Grün und Blau, da diese für die menschliche Wahrnehmung ausreichen. Somit muss obiges Gleichungssystem nur drei Mal ausgewertet werden.

2.4. Berechnung der „Vertex-Radiosity“

Da die Radiosity pro Fläche konstant ist, kann sie auf Vertices abgebildet und dann dem Renderer übergeben werden. Die Berechnung der „Vertex-Radiosities“ erfolgt nach folgendem Ansatz von Cohen und Greenberg [9]:

- Vertex-Radiosity im Inneren der Fläche wird über die angrenzenden Flächen gemittelt
- Mittelwert der Vertex-Radiosity auf der Kante und der nächstliegenden inneren Vertex-Radiosity entsprechen dem Mittelwert der Radiosity aller an dieser Vertex-Radiosity angrenzenden Flächen

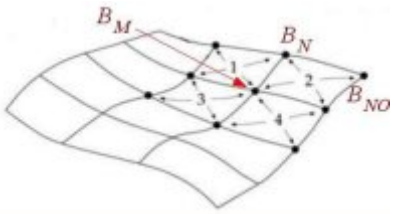


Abbildung 64. Berechnung der „Vertex-Radiosity“

Zu dieser Berechnung sei hier das Beispiel aus der Abbildung 64 in Formeln dargestellt:

$$B_M = \frac{1}{4}(B_1 + B_2 + B_3 + B_4)$$

$$\frac{1}{2}(B_N + B_M) = \frac{1}{2}(B_1 + B_2) \Rightarrow B_N = B_1 + B_2 - B_M$$

$$\frac{1}{2}(B_{NO} + B_M) = B_2 \Rightarrow B_{NO} = 2B_2 - B_M$$

2.5. Formfaktor

Der Formfaktor F_{ij} ist ein geometrischer Leitwert, der - wie schon oben erwähnt - angibt, welcher Anteil der Strahlungsleistung von der Fläche i auf die gesamte Fläche j übergeht, wobei die Form, der Flächeninhalt und die gegenseitige Lage der Flächenstücke berücksichtigt wird.

Der Formfaktor wird durch folgende Formel berechnet:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} V_{ij} dA_j dA_i \qquad (1 \leq i, j \leq n).$$

wobei

$$V_{ij} = \begin{cases} 1 & \text{falls Flächenstück } i \text{ von } j \text{ aus voll sichtbar} \\ 0 & \text{sonst} \end{cases}$$

die *Verdeckungsfunktion*, r_{ij} der Abstand zwischen den Flächenstücken i und j und θ_i der Winkel zwischen der Normalen der Fläche i und dem Richtungsvektor auf die andere Fläche ist.

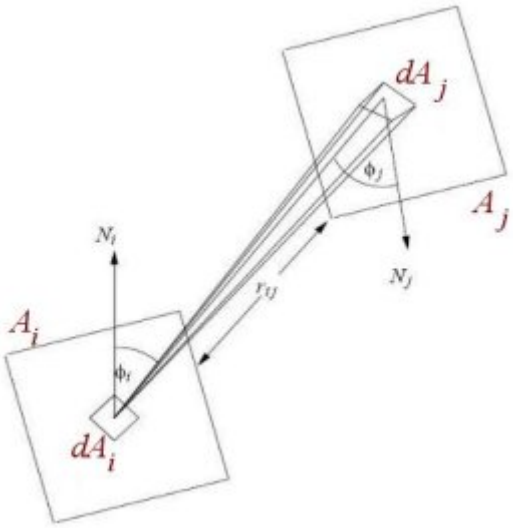


Abbildung 65. Berechnung der Formfaktoren aus der Lage der Flächen

Bemerkung 3 *Aufgrund der Definition der Formfaktoren und aufgrund der Energieerhaltung gilt folgende wichtige Eigenschaft*

$$\sum_{j=1}^{n} F_{ij} = 1 \qquad (1 \leq i \leq n).$$

Bemerkung 4 *Falls die Szene in planare Patches unterteilt wird, gilt*

$$F_{ii} = 0 \qquad \forall i.$$

Die Berechnung des Formfaktors ist der weitaus aufwendigste Teil des Radiosity-Verfahrens. Beschreibt man die Formfaktoren für endliche Flächen i und j in einer konvexen Umgebung, also wenn keine Objekte sich gegenseitig verdecken, dann fällt die Verdeckungsfunktion V_{ji} weg. Die exakte Berechnung dieses Integrals erweist sich als ziemlich schwierig. Deswegen sucht man nach alternativen Berechnungsmethoden, um das Integral ausreichend gut annnähern zu können.

2.5.1. Berechnung der Formfaktoren

Eine Möglichkeit, die Formfaktoren zu berechnen, beruht auf folgender geometrischen Lösung:

Satz 1 (Analogon von Nußelt) *Der Formfaktor von einer infinitesimal kleinen Fläche di zu einer Fläche j, gegeben durch*

$$F_{di,j} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} V_{ij} dA_j \tag{4}$$

ist äquivalent zum Betrag, den man erhält, wenn man diejenigen Teile der Fläche j, die von di aus sichtbar sind, auf eine Einheitshalbkugel projiziert, deren Zentrum sich auf di befindet, diese Projektion nochmals senkrecht auf die Grundfläche der Halbkugel projiziert und die entstehende Fläche durch die Grundfläche der Halbkugel dividiert.

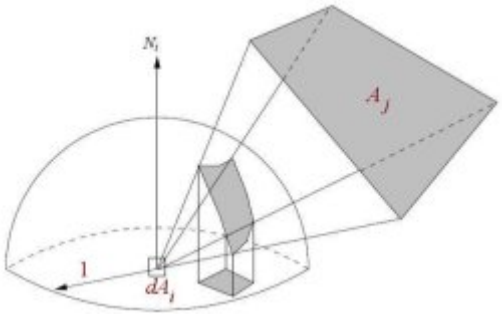


Abbildung 66. Skizze zum Analogon von Nußelt, $F_{ij} \approx F_{dA_i A_j}$

Jedoch ist auch dieses Verfahren noch sehr schwer analytisch zu beschreiben. Man approximiert aufgrund dessen die Halbkugel durch einen Halbwürfel (*Hemi-Cube-Verfahren*).

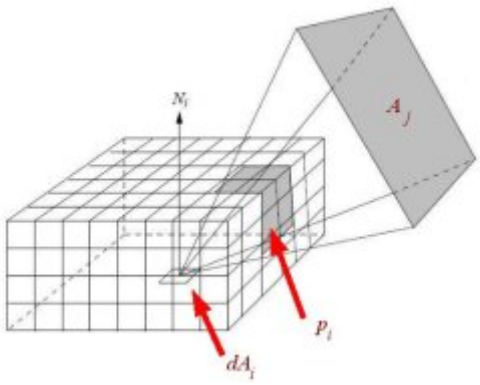


Abbildung 67. Simulation der Halbkugel durch einen Halbwürfel

Die Außenflächen des Halbwürfels sind uniform in Zellen p_i eingeteilt. Jede Zelle speichert einen Delta-Formfaktor, der von der Fläche j auf das Zentrum di projiziert wird. Der endgültige Formfaktor errechnet sich somit aus der Summe der Delta-Formfaktoren all dieser betroffenen Zellen.

$$F_{di,j} \approx \sum_i \Delta F_{p_i}$$

Eine zusätzliche Vereinfachung lässt sich erzielen, wenn man nur den "Deckel" des Würfels, also nur eine Ebene betrachtet. Dadurch verliert man einen Teil der Szeneninformation, aber der Rechenaufwand vermindert sich erheblich.
In der Abbildung 68 sind noch diverse andere Methoden aufgezeigt, wie man den Formfaktor berechnen kann.

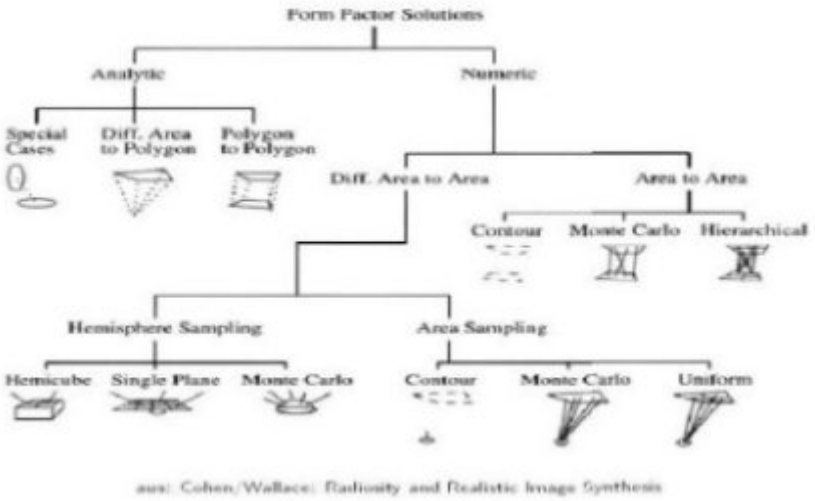


Abbildung 68. Berechnung der Formfaktoren

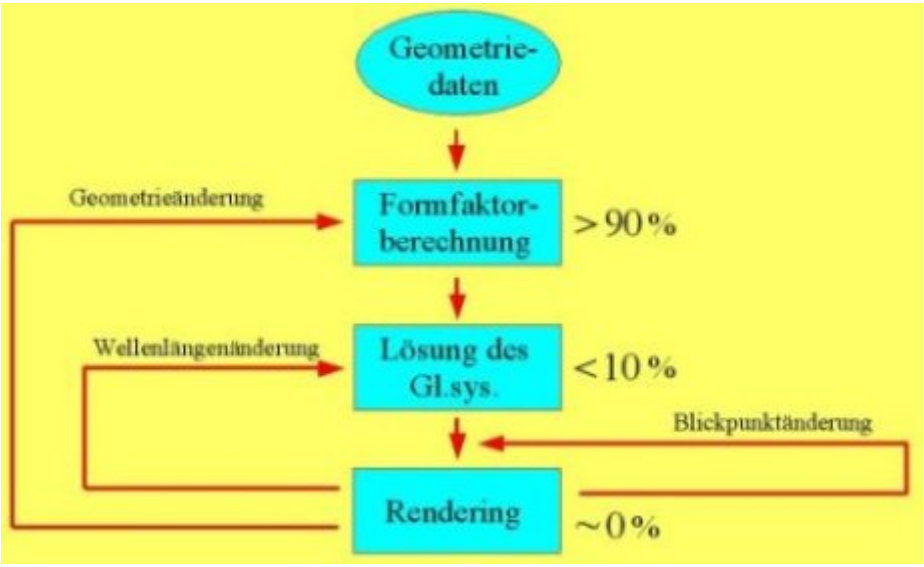


Abbildung 69. Ablauf des Radiosity-Verfahrens

2.6. Photon-Mapping

Photon-Mapping wurde 1995 von Henrik Wann Jensen entwickelt. Es ist, wie auch Radiosity, ein globales Beleuchtungsverfahren und stellt eine Erweiterung von Raytracing-basierten Methoden dar. Die Idee besteht darin, das Photonen mit einer bestimmten Energie von den Lichtquellen aus in die Szene verschossen werden. Dabei werden die Photonen an einer Oberfläche reflektiert, gestreut, gebrochen oder gespeichert. Dadurch ändert sich die Energie der Photonen. Diese Daten werden in einer sogenannten *Photon-Map* gespeichert, die dann auf diverse Weise ausgewertet werden kann, um die Szene zu rendern.

Abbildung 70. Photon-Mapping eines Innenraums (http://graphics.stanford.edu/ henrik/photonmaps, 96)

3. Lösung des Gleichungssystems und Konvergenzsätze

3.1. Direkte Verfahren

Im Folgenden soll nun gezeigt werden, wie man das Gleichungssystem (3), in Kurzschreibweise $AB = E$, nach B auflösen kann.

3.1.1. Gaußelimination

Am naheliegensten wäre die Gaußelimination, jedoch ist sie nur bei vollbesetzten $n \times n$ Matrizen sinnvoll, da es bei dünnbesetzten Matrizen (also Matrizen, deren Anzahl der Nicht-Null-Einträge wesentlich kleiner als n^2 sind) zu so genannten "Fill-Ins" kommt. Desweiteren spricht auch der hohe Rechenaufwand von $O(n^3)$ gegen diese Methode.

3.1.2. Invertierung

Die Invertierung der Matrix A, also das Gleichungssystem $B = A^{-1}E$ zu lösen, ist auch eine Möglichkeit der Lösung. Jedoch ist dies bei zu großen Matrizen (also 1000×1000) unpraktisch, da der Aufwand auch bei $O(n^3)$ liegt.

3.2. Iterationsverfahren

Eine andere bessere Möglichkeit der Lösung sind die Iterationsverfahren:

3.2.1. Allgemeine lineare Iterationsverfahren

Gegeben sei eine nichtsinguläre $n \times n$-Matrix A und ein lineares Gleichungssystem

$$Ax = b$$

mit der exakten Lösung $x = A^{-1}b$. Ausgehend von einem Startvektor $x^{(0)}$ wird eine Folge von Vektoren $x^{(0)} \to x^{(1)} \to x^{(2)} \to \ldots$ erzeugt, die gegen die gesuchte Lösung x konvergiert, d.h. es werden Fixpunktverfahren der Form

$$x^{(i+1)} = \Phi(x^{(i)}) \qquad (i = 0, 1, \ldots)$$

betrachtet, wobei eine Iterationsfunktion Φ so konstruiert wird, dass sie genau einen Fixpunkt besitzt und dieser gerade die gesuchte Lösung $x = A^{-1}b$ ist.

Durch Hinzunahme einer beliebigen nichtsingulären $n \times n$-Matrix M erhält man eine solche Iterationsvorschrift aus der Gleichung

$$Mx + (A - M)x = b, \tag{5}$$

indem man

$$Mx^{(i+1)} + (A - M)x^{(i)} = b \tag{6}$$

setzt und nach $x^{(i+1)}$ auflöst

$$x^{(i+1)} = x^{(i)} - M^{-1}(Ax^{(i)} - b) = (I - M^{-1}A)x^{(i)} + M^{-1}b. \tag{7}$$

Definition 1 *Ein Iterationsverfahren zur Lösung von $Ax = b$ heißt konsistent $:\Leftrightarrow$ x ist Fixpunkt der Iteration.*

Zur Vereinfachung wird $S := (I - M^{-1}A)$ gesetzt. Bevor das erste Konvergenzkriterium behandelt wird, werden noch der Spektralradius und die Matrixnorm definiert.

Definition 2 (Spektralradius) *Sei $A \in \mathbb{C}^{n\times n}$ eine beliebige Matrix. Mit $\rho(A) := \max\limits_{1\le i\le n} |\lambda_i|$ wird der so genannte Spektralradius definiert, wobei die λ_i die Eigenwerte von A sind.*

Definition 3 (Matrixnorm) *Sei $A \in \mathbb{C}^{n\times n}$ eine beliebige Matrix. Für $x \in \mathbb{C}^n$ und einer gegebenen Vektornorm $\|x\|$ wird die Matrixnorm definiert durch $|||A||| := \sup\limits_{x\neq 0} \frac{\|Ax\|}{\|x\|}$.*

Es gilt nun das folgende Konvergenzkriterium:

Satz 2 ***1.*** *Das Verfahren* (7) *ist genau dann konvergent, wenn*

$$\rho(S) < 1.$$

2. *Hinreichend für die Konvergenz von* (7) *ist die Bedingung*

$$|||S||| < 1.$$

Es sei hier erwähnt, dass dies der einzige Beweis ist, der in der Ausarbeitung detailliert beschrieben wird. Alle anderen Beweise sind im Anhang aufgezeichnet.
Zum Beweis dieses Satzes benötigt man nun folgende zwei Sätze:

Satz 3 (Hirsch) *Für alle Eigenwerte* λ *von A gilt*

$$|\lambda| \leq |||A|||.$$

Satz 4 ***1.*** *Zu jeder Matrix* A *und jedem* $\epsilon > 0$ *existiert eine Vektornorm mit*

$$|||A||| \leq \rho(A) + \epsilon.$$

2. *Hat jeder Eigenwert* λ *von A mit der Eigenschaft* $|\lambda| = \rho(A)$ *nur lineare Elementarteiler, so existiert sogar eine Vektornorm mit*

$$|||A||| = \rho(A).$$

Beweis von Satz 2:

Für den Fehler $f_i := x^{(i)} - x$ folgt durch Subtraktion der Gleichung (5) von (6):

$$f_{i+1} = Sf_i$$

bzw.

$$f_i = S^i f_0 \qquad\qquad i = 0, 1, \ldots \tag{8}$$

$=>$:
Sei nun (7) konvergent. Dann ist $\lim\limits_{i\to\infty} f_i = 0$ für alle f_0. Wählt man f_0 als Eigenvektor zum Eigenwert λ von S, dann folgt mit (8) daraus

$$f_i = \lambda^i f_0. \tag{9}$$

Und schließlich folgt $|\lambda| < 1$, da $\lim_{i\to\infty} f_i = 0$ und daraus $\rho(S) < 1$.

$<=$:

Sei umgekehrt $\rho(S) < 1$, so folgt aus Satz 4 sofort $\lim_{i\to\infty} S^i = 0$ und so $\lim_{i\to\infty} f_i = 0$ für alle f_0. Der zweite Teil des Satzes folgt unmittelbar aus Satz 3.

□

Aus diesem Satz kann man vermuten, das das Verfahren umso schneller konvergiert, je kleiner $\rho(S)$ ist und umso schlechter, je näher der Wert von $\rho(S)$ an der 1 liegt. Folgender Satz sagt Genaueres.

Satz 5 *Für das Verfahren* (7) *gilt:*

$$\sup_{f_0\neq 0} \limsup_{i\to\infty} \sqrt[i]{\frac{\|f_i\|}{\|f_0\|}} = \rho(S)$$

für die Fehler $f_i = x^{(i)} - x$. *Dabei ist* $\|\cdot\|$ *eine beliebige Norm.*

Es werden nun spezielle Verfahren erläutert, die von der Wahl der Matrix M abhängen. Dabei soll nun im Folgenden die Zerlegung $A = L + D + R$ gelten, wobei $D = diag(a_{11}, \ldots, a_{nn})$ und

$$L = \begin{pmatrix} 0 & \ldots & \ldots & 0 \\ a_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \ldots & a_{n,n-1} & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 0 & a_{12} & \ldots & a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ 0 & \ldots & \ldots & 0 \end{pmatrix}.$$

Die Matrix M sollte nun eine "leicht zu invertierende" Matrix sein und es sollte gelten $M \approx A$, denn dann würde das Verfahren die exakte Lösung liefern.

3.2.2. *Jacobi-Verfahren (Gesamtschrittverfahren)*

Wählt man nun für $M = D$, so resultiert daraus das Jacobi-Verfahren mit der Vorschrift

$$\begin{aligned} x^{(i+1)} &= (I - D^{-1}A)x^{(i)} + D^{-1}b \\ &= -D^{-1}(L + R)x^{(i)} + D^{-1}b. \end{aligned}$$

3.2.3. *Gauss-Seidel-Verfahren (Einzelschrittverfahren)*

Für das Gauss-Seidel-Verfahren wählt man $M = L + D$ und die dazugehörige Iterationsvorschrift

$$\begin{aligned} x^{(i+1)} &= (I - (D + L)^{-1}A)x^{(i)} + (D + L)^{-1}b \\ &= -(D + L)^{-1}Rx^{(i)} + (D + L)^{-1}b. \end{aligned}$$

Im Weiteren wird zunächst die Konvergenz des Jacobi-Verfahrens betrachtet. Dazu definieren wir zunächst den Graphen, die Irreduzibilität einer Matrix sowie dieDiagonaldominanz.

Definition 4 (Graph) *Sei $A \in \mathbb{K}^{I\times I}$ zur Indexmenge I. Dann wird der Graph $G(A)$ der Matrix A als die folgende Teilmenge aller Paare aus $I \times I$ bezeichnet:*

$$G(A) := \{(\alpha, \beta) \in I \times I : a_{\alpha,\beta} \neq 0\}$$

Definition 5 (Irreduzibilität) *$A \in \mathbb{K}^{I\times I}$ heißt irreduzibel, wenn jedes $\alpha \in I$ mit jedem $\beta \in I$ verbunden ist. Sonst ist sie reduzibel.*

Definition 6 (Diagonaldominanz) *Eine Matrix $A \in \mathbb{K}^{I\times I}$ heißt stark diagonaldominant, wenn*

$$|a_{\alpha\alpha}| > \sum_{\beta\in I,\beta\neq\alpha} |a_{\alpha\beta}| \qquad \forall\alpha \in I$$

Eine Matrix $A \in \mathbb{K}^{I\times I}$ heißt schwach diagonaldominant, wenn

$$|a_{\alpha\alpha}| \geq \sum_{\beta\in I,\beta\neq\alpha} |a_{\alpha\beta}| \qquad \forall\alpha \in I$$

Eine Matrix $A \in \mathbb{K}^{I\times I}$ heißt irreduzibel diagonaldominant, wenn A irreduzibel und schwach diagonaldominant ist, und außerdem gilt, dass

$$|a_{\alpha\alpha}| > \sum_{\beta\in I,\beta\neq\alpha} |a_{\alpha\beta}| \qquad \textit{für mindestens ein } \alpha \in I.$$

Es gilt nun folgender Konvergenzsatz für das Jacobi-Verfahren:

Satz 6 *Das Jacobi-Verfahren konvergiert für alle stark diagonaldominante Matrizen A.*

Man kann die Forderung sogar noch einschränken.

Satz 7 *Das Jacobi-Verfahren konvergiert für alle irreduzibel diagonaldominante Matrizen A.*

Für das Gauss-Seidel-Verfahren gilt:

Satz 8 *Das Gauss-Seidel-Verfahren konvergiert für alle stark diagonaldominante Matrizen A und es gilt*

$$\left\|\left|S_G\right|\right\|_\infty \leq \left\|\left|S_J\right|\right\|_\infty < 1.$$

Satz 9 *Das Gauss-Seidel-Verfahren konvergiert für alle irreduzibel diagonaldominante Matrizen A.*

Im Folgenden soll nun gezeigt werden, dass die Radiosity-Matrix A aus Gleichung (3) auch wirklich diagonaldominant ist. Sei $a_{ij} \in A$ $(1 \leq i, j \leq n)$. Man betrachtet nun die Diagonalelemente der Matrix A:

$$a_{ii} = 1 - \rho_i F_{ii} = \sum_{j=1}^{n} F_{ij} - \rho_i F_{ii} > \sum_{j=1}^{n} \rho_i F_{ij} - \rho_i F_{ii} = \sum_{j=1, j\neq i}^{n} \rho_i F_{ij},$$

$$\text{da} \quad \sum_{j=1}^{n} F_{ij} = 1 \quad \text{und} \quad F_{ij} > \rho_i F_{ij} \ (\rho_i < 1)$$

$$\Rightarrow \ |a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

Also sind beide Verfahren auf die Radiosity-Matrix anwendbar. Es sollen nun die Unterschiede der beiden Verfahren untersucht werden.

3.3. Jacobi vs Gauss-Seidel

Das Jacobi-Verfahren benötigt in jeder Iteration zwei Vektoren, da der neue immer aus dem alten berrechnet wird. Das Gauss-Seidel-Verfahren braucht dagegen nur einen Vektor, da es in jedem Iterationsschritt sofort die bis dahin errechneten Werte für die weitere Berechnung benutzt. D.h. das Jacobi-Verfahren errechnet immer nur eine Reflexion pro Iteration, während das Gauss-Seidel-Verfahren mehrere Reflexionen pro Iteration berechnet. Das ist auch der Grund, warum das Gauss-Seidel-Verfahren im allgemeinen fast doppelt so schnell konvergiert wie das Jacobi-Verfahren.

In den Abbildungen 71 und 72 werden die Geschwindigkeitsunterschiede zwischen beiden Verfahren dargestellt.

	Jacobi		Gauss-Seidel	
Dimension / Type	% Convergences	Avg. Iterations	% Convergences	Avg. Iterations
3x3 / Random	12.1%	114.7	19.0%	67.0
4x4 / Random	2.1%	307.9	4.4%	100.6
5x5 / Random	0.2%	2403.5	0.5%	277.0
3x3 / Tridiagonal	32.2%	57.3	32.2%	29.5
4x4 / Tridiagonal	17.4%	89.2	17.4%	46.9
5x5 / Tridiagonal	11.7%	233.8	11.8%	158.3

Abbildung 71. Konvergenz und Anzahl der Iterationen bei einer Probe mit 1000 Zufallsmatrizen und einer Genauigkeit von 10^{-4}

Das Programm, das die Werte für obige Tabelle errechnet hat, kann man auf der Seite http://www.leosingleton.com/projects/courses/math2601/lineq/ herunterladen.

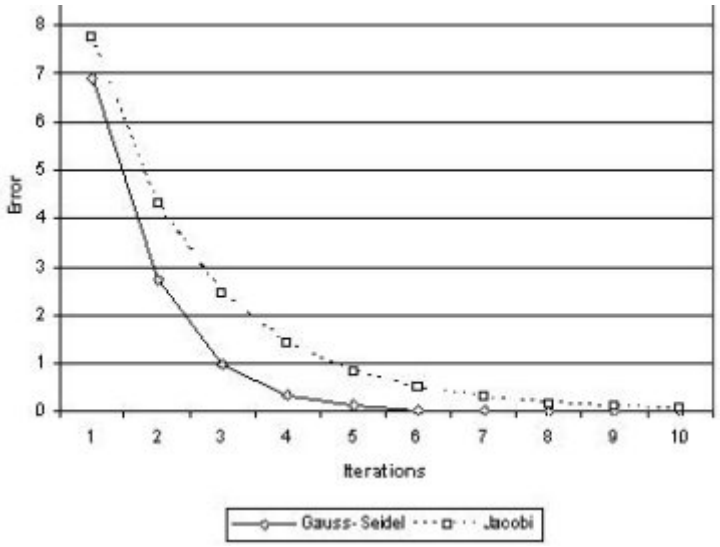


Abbildung 72. Gaus-Seidel- und Jacobiverfahren im Vergleich

3.4. Gathering vs Shooting

3.4.1. Gathering

Jacobi- und Gauss-Seidel-Verfahren sind so genannte Gathering-Methoden(siehe [10]). Damit ist gemeint, dass ein Patch die Radiosity der übrigen Patches in der Szene *einsammelt*. Die Lösung einer Zeile des Gleichungssystems beim Gauss-Seidel-Verfahren liefert den Radiosity-Wert eines Patches. Genauer: Gathering über *einen* Hemi-Cube erlaubt es die Radiosity über *einen* Patch zu aktualisieren.

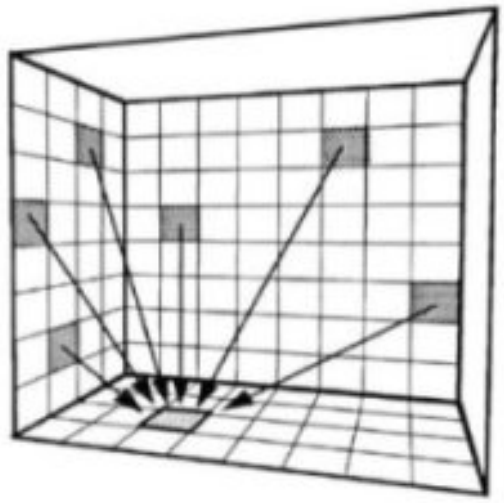


Abbildung 73. Gathering

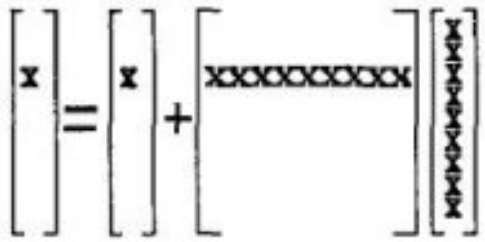


Abbildung 74. Matrizenbelegung im Fall des Gathering, $B_i = E_i + \sum_{j=1}^{n}(\rho_i F_{ij})B_j$

```
for (i = 0; i < n; i++)
      B[i] = E[i];

while (no convergence)
{
   for (i = 0; i < n; i++)
   {
      B_sum = 0;
      for (j = 0; j < n; j++)
         B_sum += F[i][j] * B[j];

      B[i] = E[i] + rho[i] * B_sum;
   }
   render(B);
}
```

Abbildung 75. Pseudo-Code für das Gathering

3.4.2. Shooting

Zu den Shooting-Verfahren(siehe [10]) gehört die Southwell-Methode. Beim Shooting wird jeweils das Patch mit der höchsten Energie in die Umgebung *verschossen*. Genauer: Shooting über *einen* Hemi-Cube erlaubt es, die Radiosity *mehrerer* Patches zu aktualisieren.

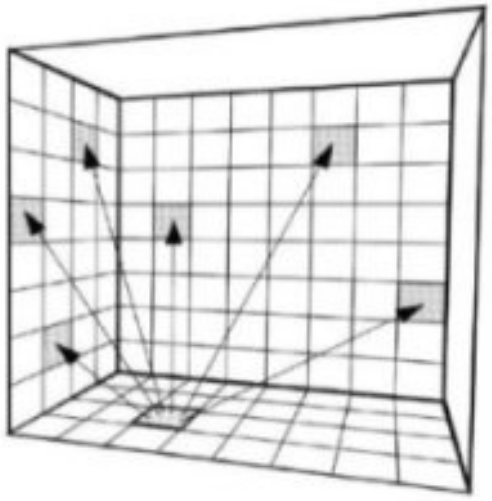


Abbildung 76. Shooting

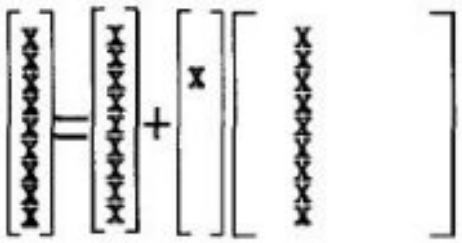


Abbildung 77. Matrizenbelegung im Fall des Shooting, $B_j = B_j + B_i(\rho_j F_{ji})$

```
for (i = 0; i < n; i ++)
      B[i] = dB[i] = E[i];       //dB[i] : unshot radiosity

while (no convergence)
{
   set i as dB[i] is the largest
   {
   for (j = 0; j < n; j ++)
      {
      db = rho[j] * F[j][i] * dB[i];
      dB[j] += db;                /*update change since last
                                     time patch j shot light*/
       B[j] += db;                /*update total radiosity of
                                     patch j*/
      }
      dB[i] = 0;                  /*reset unshot radiosity for
                                     patch i to zero*/
   }
   render (B);
}
```

Abbildung 78. Pseudo-Code für das Shooting

3.4.3. Progressive Refinement

Hier sind diverse Ausschnitte aus dem Paper [10], in denen jeweils nach 1, 2, 24 und 100 Iterationsschritten Aufnahmen der Szene gemacht worden sind.

Bei der Benutzung von einfachem Gauss-Seidel-Verfahren bleibt die Szene auch nach 100 Schritten relativ dunkel.

Abbildung 79. Gauss-Seidel nur mit Gathering-Verfahren

Hier wurde nur das Shooting ohne die Sortierung nach dem größten Energiewert verwendet, ein nicht allzu großer Unterschied zum Gathering.

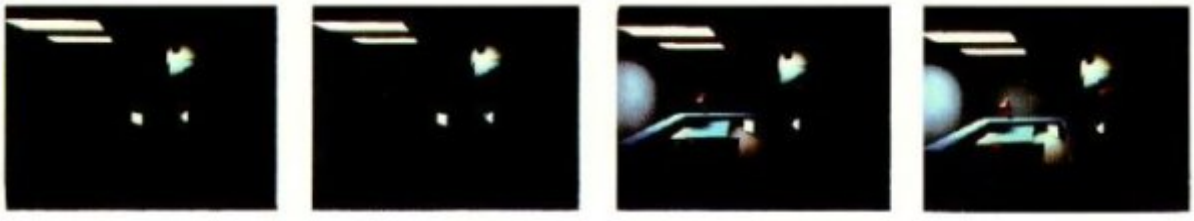


Abbildung 80. Gauss-Seidel nur mit Shooting-Verfahren

Wenn man beim Shooting-Verfahren zusätzlich nach der Helligkeit der auftretenden Patches sortiert,

erhellt sich auch die Szene schneller. Man erkennt deutlich den Unterschied zum reinen Shooting-Verfahren.

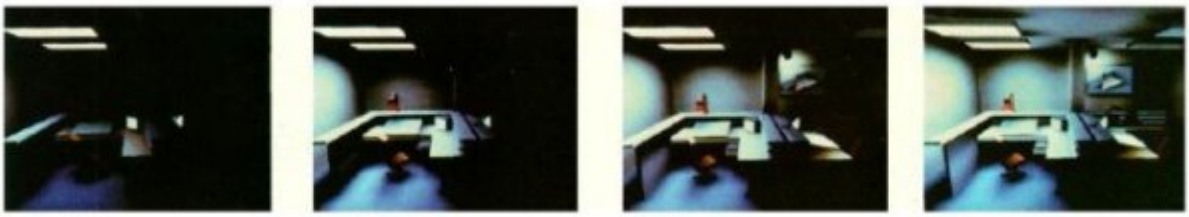


Abbildung 81. Kombination aus Shooting mit Sortierverfahren

Hier wurde auch noch ein konstanter ambienter Anteil von Anfang an aus Sichtbarkeitsgründen in die Szene eingerechnet. Er hängt in jedem Iterationsschritt von den jeweils bis dahin berechneten Radiosity-Werten aller Patches und der Reflektivität der Umgebung ab. Er geht also nicht in die Lösung des Gleichungssystems ein und nimmt mit jedem Iterationsschritt nach und nach ab.

Abbildung 82. shooting, sorting and ambient

In Abbildung 83 sind die einzelnen Konvergenzkurven der jeweiligen Verfahren im Diagramm dargestellt.

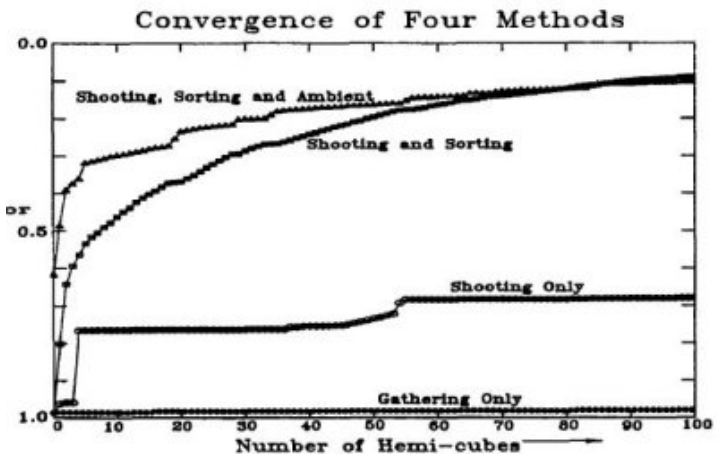


Abbildung 83. Konvergenz der verschiedenen Methoden

4. Bilder und Anwendungen

Indirekte Beleuchtung und Lichtführung sind besonders in Museen gefragt, wo die Exponate gleichmäßig ausgeleuchtet sein sollen. Für derart realistische Simulationen ist das Radiosity-Verfahren einzig tauglich.

Ein sehr bekanntes Beispiel findet sich auf der Webseite der Graphikgruppe an derCornell University (siehe Abbildung 84).

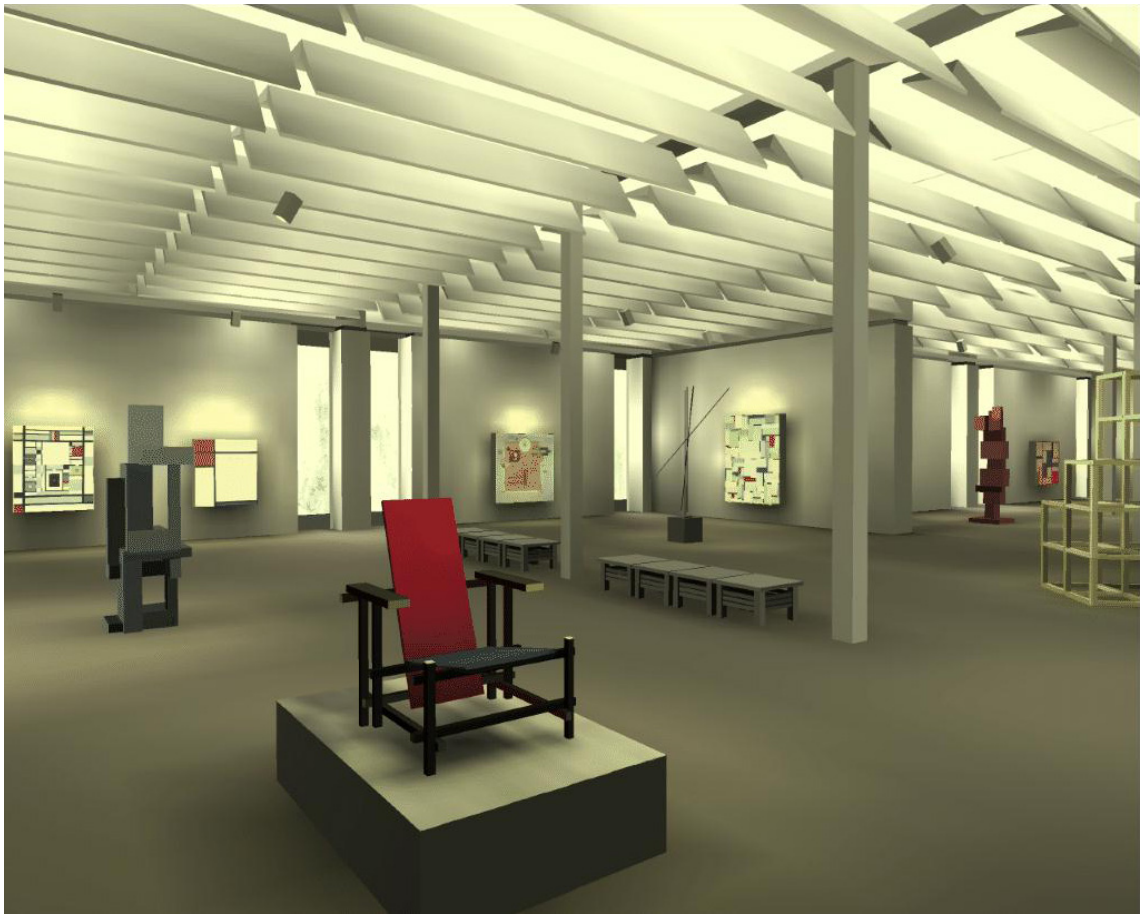


Abbildung 84. Museum (Cornell University)

In Abbildung 85 ist die so genannte *Cornell Box* dargestellt, die als Benchmark für das Lösen der Radiosity-Gleichung dient. Um realistische Bilder zu erzeugen, müssen die einzelnen Farbkanäle gesondert behandelt werden. Dadurch wird der als *Colorbleeding* bekannte Effekt erzielt: farbige Wände strahlen ihre Farbe auf hellere Objekte ab.

Abbildung 85. Cornell Box (2370 Patches, mittels Gouraud-Shading gerendert, „Colorbleeding“)

Radiosity wird hauptsächlich im Innenarchitekturbereich verwendet, um möglichen Kunden eine digitale Führung durch ihr zukünftiges Heim zu ermöglichen. De folgenden Abbildungen 86, 87, 88 und 89 stammen sämtlich von *3d-architectural-rendering* (www.archiform3d.com).

Abbildung 86. Wohnbereich eines Appartments

Abbildung 87. Essbereich eines Appartments

Abbildung 88. Küchenzeile

Abbildung 89. Außenansicht der Wohnanlage

Literatur

[1] M. F. Cohen et al., *Radiosity and realistic image synthesis*, Academic Press Professional, San Diego 1993, ISBN 0-12-178270-0

[2] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips *Grundlagen der Computergraphik*, Addison-Wesley, 1994, pp. 583-590

[3] Josef Pöpsel, Ute Claussen, Rolf-Dieter Klein, Jürgen Plate *Computergrafik*, Springer Verlag, 1994, pp. 151-172, 189-200

[4] Stoer, Bulirsch, *Numerische Mathematik* 2, Springer-Lehrbuch, 1990, pp. 256-266

[5] Christian Kanzow, *Numerik linearer Gleichungssysteme - Direkte und iterative Verfahren*, Springer Verlag, 2005, pp. 139-149.

[6] Robert Schaback, Holger Wendland, *Numerische Mathematik*, Springer Verlag, 2005, pp. 93-101

[7] Peter Deuflhard, Andreas Hohmann, *Numerische Mathematik* 1, de Gruyter Lehrbuch, 2002, pp. 263-269

[8] Goral, C.M.; Torrance, D.E.; Greenberg, D.P.; Battaile, G., *Modeling the Interaction of Light Between Diffuse Surfaces*, Proceedings of SIGGRAPH 84, 1984, S. 213-222

[9] Cohen, M.F.; Greenberg, D.P., *The Hemi-Cube: A Radiosity Solution for Complex Environments*, Proceedings of SIGGRAPH 85, 1988, S. 31-40

[10] Cohen, M.F.; Chen S.E.; Wallce, J.R.; Greenberg, D.P., *A Progressive Refinement Approach to Fast Radiosity Image Generation*, Proceedings of SIGGRAPH 88, 1988, S. 75-84

[11] Bresink, Marcel, *Deutschsprachige Terminologie des Radiosity-Verfahrens*, Universität Koblenz-Landau, 29/97

A Weitere Iterationsverfahren

1.1. Richardson-Verfahren

Die am besten zu invertierende Matrix wäre ohne Zweifel ein Vielfaches der Einheitsmatrix $M = \tau \cdot I$
Das so entstandene Verfahren

$$x^{(i+1)} = x^{(i)} - \frac{1}{\tau} \cdot Ax^{(i)} + \frac{1}{\tau} \cdot b = (I - \frac{1}{\tau} \cdot A)x^{(i)} + \frac{1}{\tau} \cdot b$$

wird Richardson-Verfahren genannt.

1.2. Nachiterations-Verfahren

Hier handelt es sich um einen besonderen Spezialfall. Als Resultat eines Eliminationsverfahrens zur Lösung von $Ax = b$ erhält man infolge von Rundungsfehlern eine relativ gute Näherungslösung $x^{(0)}$ für die exakte Lösung x und eine untere Dreiecksmatrix $\widetilde{L}$ bzw. obere Dreiecksmatrix $\widetilde{R}$ mit $\widetilde{L}\widetilde{R} \approx A$. Mit der Näherungslösung $x^{(0)}$ als Startvektor und mit $M := \widetilde{L}\widetilde{R}$ wird dann nachiteriert. Daraus folgt aus Gleichung (2):

$$x^{(i+1)} = x^{(i)} + u^{(i)}$$

wobei $u^{(i)} := \widetilde{R}^{-1}\widetilde{L}^{-1}r^{(i)}$ und $r^{(i)} := b - Ax^{(i)}$ das Residuum ist.
Es ist zu beachten das $u^{(i)}$ relativ leicht aus den gestaffelten Gleichungssystemen

$$\widetilde{L}t = r^{(i)}, \qquad \widetilde{R}u^{(i)} = t,$$

berechnet werden kann. Das Verfahren konvergiert relativ schnell, jedoch kann bei Berechnung der Residuen Auslöschung auftreten. Aus diesem Grund berechnet man die Residuen $r^{(i)}$ mit doppelter Genauigkeit.

1.3. SOR-Verfahren (Successive Overrelaxation) bzw. Relaxationsverfahren

Eine weitere Möglichkeit, bessere Konvergenzbedingungen, als mit dem Gauss-Seidel-Verfahren zu erzielen, ist eine ganze Klasse von Matrizen $M(\omega)$ in Abhängigkeit eines Partameters ω zu betrachten. Die Kunst liegt nun darin, ω so zu wählen, dass $\rho(I - M(\omega)^{-1}A)$ möglichst klein wird. Man wählt $M(\omega)$ folgendermaßen:

$$M(\omega) = \frac{1}{\omega}D(I + \omega L)$$

Man kann jedoch beweisen, dass das Verfahren nur für $0 < \omega < 2$ konvergiert und $\rho(I - M(\omega)^{-1}A)$ minimal wird, wenn

$$\omega = \frac{2}{2 - \lambda_{\min} - \lambda_{\max}}$$

gewählt wird. Für $\omega < 1$ spricht man von Unterrelaxation und für $\omega > 1$ von Überrelaxation. Jedoch gelten diese Sätze nur, falls A positiv definit ist.

B Beweise

Beweis von Satz 3:

Ist x Eigenvektor zum Eigenwert λ, so folgt aus

$$Ax = \lambda x, \qquad\qquad x \neq 0$$

die Beziehung

$$\|\lambda x\| = |\lambda| \cdot \|x\| \leq \|\|A\|\| \cdot \|x\|\,,$$
$$|\lambda| \leq \|\|A\|\|\,.$$

□

Beweis von Satz 4:

Sei nun die Maximumnorm bzgl. der Matrixnorm definiert durch

$$\|\|A\|\|_\infty = \max_i \sum_k |a_{ik}|.$$

1. Zu A existiert eine nichtsinguläre Matrix T, so dass

$$J = TAT^{-1}$$

die Jordansche Normalform von A ist, d. h. J besteht aus Diagonalblöcken der Gestalt

$$C_\nu(\lambda_i) = \begin{pmatrix} \lambda_i & 1 & & & 0 \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ 0 & & & & \lambda_i \end{pmatrix}.$$

wobei $1 \leq \nu \leq n$. Durch die Transformation $J \to D_\epsilon^{-1} J D_\epsilon$ mit der Diagonalmatrix

$$D_\epsilon := diag(1, \epsilon, \epsilon^2, \ldots, \epsilon^{n-1}), \qquad\qquad \epsilon > 0$$

erreicht man, dass die $C_\nu(\lambda_i)$ übergehen in

$$\begin{pmatrix} \lambda_i & \epsilon & & & 0 \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & \epsilon \\ 0 & & & & \lambda_i \end{pmatrix}$$

Daraus folgt

$$\left\|\|D_\epsilon^{-1} J D_\epsilon\|\right\|_\infty = \left\|\|D_\epsilon^{-1} T A T^{-1} D_\epsilon\|\right\|_\infty \leq \rho(A) + \epsilon.$$

Es gilt nun allgemein: Ist S eine nichtsinguläre Matrix, $\|\cdot\|$ eine Vektornorm, so ist auch $p(x) := \|Sx\|$ eine Vektornorm und es ist $\|\|A\|\|_p = \|\|SAS^{-1}\|\|$. Für die Norm $p(x) := \|D_\epsilon^{-1}Tx\|_\infty$ folgt daraus

$$\|\|A\|\|_p = \left\|\|D_\epsilon^{-1}TAT^{-1}D_\epsilon\|\right\|_\infty \leq \rho(A) + \epsilon.$$

2. Die Eigenwerte λ_i von A seien wie folgt geordnet

$$\rho(A) = |\lambda_1| = \ldots = |\lambda_s| > |\lambda_{s+1}| \geq \ldots \geq |\lambda_n|.$$

Dann hat nach Voraussetzung für $1 \leq i \leq s$ jedes Jordankästchen $C_\nu(\lambda_i)$ in J die Dimension 1, d.h. $C_\nu(\lambda_i) = [\lambda_i]$. Wählt man

$$\epsilon = \rho(A) - |\lambda_{s+1}|,$$

so gilt deshalb

$$\left\|\|D_\epsilon^{-1}TAT^{-1}D_\epsilon\|\right\|_\infty = \rho(A).$$

Für die Norm $p(x) := \|D_\epsilon^{-1}Tx\|_\infty$ folgt wie in **1**

$$\|\|A\|\|_p = \rho(A).$$

□

Beweis von Satz 5:

Sei $\|\cdot\|$ eine beliebige Norm und $\|\|\cdot\|\|$ die zugehörige Matrixnorm. Nach Gleichung (8) und (9) aus dem Beweis von Satz 2 folgt sofort, indem man für f_0 die Eigenvektoren für S wählt, dass

$$\sup_{f_0 \neq 0} \limsup_{i \to \infty} \sqrt[i]{\frac{\|f_i\|}{\|f_0\|}} \geq \rho(S).$$

Sei nun $\epsilon > 0$ beliebig. Dann gibt es nach Satz 4 eine Vektornorm $N(\cdot)$, so daß für die zugehörige Matrixnorm $\|\|\cdot\|\|_N$ gilt

$$\|\|S\|\|_N \leq \rho(S) + \epsilon.$$

Man kann beweisen, dass alle Normen auf dem $\mathbb{C}^n$ äquivalent sind und dass es Konstanten $m, M > 0$ gibt mit

$$m \cdot \|x\| \leq N(x) \leq M \cdot \|x\|.$$

Ist nun $f_0 \neq 0$ beliebig, so folgt aus diesen Ungleichungen und Gleichung 8:

$$\begin{aligned}\|f_i\| &\leq \frac{1}{m} N(f_i) = \frac{1}{m} N(S^i f_0) \\ &\leq \frac{1}{m} (\|\|S\|\|_N)^i N(f_0) \\ &\leq \frac{M}{m} (\rho(S) + \epsilon)^i \|f_0\|\end{aligned}$$

bzw.

$$\sqrt[i]{\frac{\|f_i\|}{\|f_0\|}} \leq (\rho(S) + \epsilon) \sqrt[i]{\frac{M}{m}}.$$

Wegen $\lim_{i\to\infty} \sqrt[i]{\frac{M}{m}} = 1$ erhält man $\sup_{f_0\neq 0} \limsup_{i\to\infty} \sqrt[i]{\frac{\|f_i\|}{\|f_0\|}} \leq \rho(S) + \epsilon$ und da $\epsilon > 0$ beliebig war, folgt die Behauptung.

□

Beweis von Satz 6:

Aus der starken Diagonaldominanz von A folgt zunächst $a_{ii} \neq 0$ für alle $i = 1, \ldots, n$, so dass das Jacobi-Verfahren durchführbar ist.

$$\||S_J|\|_\infty = \big\||I - D^{-1}A|\big\|_\infty = \big\||-D^{-1}(L+R)|\big\|_\infty = \max_{i=1,\ldots,n} \frac{1}{|a_{ii}|} \sum_{j=1, j\neq i}^{n} |a_{ij}| < 1.$$

Die Behauptung folgt dann aus Satz 2.

□

Beweis von Satz 7:

Da jede irreduzibel diagonaldominante Matrix regulär ist (ohne Beweis), sind die $a_{ii} \neq 0\ \forall i$, denn sonst wären alle Einträge von A gleich Null, was ein Widerspruch zur Regularität wäre. Sei nun $\lambda \in \mathbb{C}$ mit $|\lambda| \geq 1$ beliebig gegeben. Da die Matrix A nach Voraussetzung irreduzibel ist, gilt dies auch für Matrizen $L + R, S_J = -D^{-1}(L + R)$ und $B := S_J - \lambda I$. Für $B = S_J - \lambda I$ ist wegen der Diagonaldominanz

$$\sum_{j=1, j\neq i}^{n} |b_{ij}| = \sum_{j=1, j\neq i}^{n} |\frac{a_{ij}}{a_{ii}}| \leq^* 1 \leq |\lambda| = |b_{ii}| \qquad \forall i = 1, \ldots, n\ ,$$

wobei in $(*)$ für mindestens einen Index $i_0 \in \{1, \ldots, n\}$ die starke Ungleichung erfüllt ist. Folglich ist B irreduzibel diagonaldominant, also insbesondere regulär. Da aber B für jeden Eigenwert λ von S_J singulär sein muss, folgt daraus, dass $\rho(S_J) < 1$ und somit die Behauptung.

□

Beweis von Satz 8:

$\||S_J|\|_\infty < 1$ wurde schon im Beweis von Satz 6 gezeigt. Bleibt also noch $\||S_G|\|_\infty \leq \||S_J|\|_\infty$ zu zeigen. Gemäß Definition gilt:

$$\||S_G|\|_\infty = \max_{\|x\|_\infty = 1} \|S_G x\|_\infty\ .$$

Sei daher $x \in \mathbb{R}^n$ mit $\|x\|_\infty = 1$ beliebig gegeben. Aus der Definition von S_G folgt für $y := S_G x$

$$y = \frac{1}{a_{ii}} \left(-\sum_{j=1}^{i-1} a_{ij} y_j - \sum_{j=i+1}^{n} a_{ij} x_j \right) \qquad \forall i = 1, \ldots, n.$$

Sei nun $\kappa := \| |S_J| \|_\infty$ Man zeigt per Induktion, dass $|y_i| \leq \kappa$.
Für $i = 1$ ist

$$|y_1| \leq \frac{1}{|a_{11}|} \left(\sum_{j=2}^{n} |a_{1j}||x_j| \right) \leq \frac{1}{|a_{11}|} \sum_{j=2}^{n} |a_{ij}| \underbrace{\|x\|_\infty}_{=1} \leq \| |S_J| \|_\infty = \kappa.$$

Gilt bereits $|y_i| \leq \kappa$. für die Komponenten $j = 1, \ldots, i-1$, so folgt

$$|y_i| \leq \frac{1}{|a_{ii}|} \left(\sum_{j=1}^{i-1} |a_{ij}| \cdot \underbrace{|y_j|}_{\leq \kappa} + \sum_{j=i+1}^{n} |a_{ij}| \cdot \underbrace{|x_j|}_{\leq \|x\|_\infty} \right)$$

$$\leq \frac{1}{|a_{ii}|} \left(\sum_{j=1}^{i-1} |a_{ij}| \cdot \underbrace{\kappa}_{\leq 1} + \sum_{j=i+1}^{n} |a_{ij}| \cdot \underbrace{\|x|\|_\infty}_{=1} \right)$$

$$\leq \frac{1}{|a_{ii}|} \left(\sum_{j=1}^{i-1} |a_{ij}| + \sum_{j=i+1}^{n} |a_{ij}| \right)$$

$$\leq \| |S_J| \|_\infty = \kappa$$

Also ist $\|y\|_\infty \leq \kappa$ und somit $\| |S_G| \|_\infty \leq \kappa$. Daraus folgt die Behauptung.

□

Beweis von Satz 9:

Der Beweis läuft analog wie der Beweis zu Satz 7.
Man betrachtet $S_G - \lambda I = -(D + L)^{-1}(R + (\lambda D + \lambda L))$, definiert $B := R + \lambda D + \lambda L$ und geht dann analog vor.

□

Abbildungsverzeichnis