Ray Tracing mit Octrees

Markus Kirschmann *
Universität Ulm

Zusammenfassung

Quadtrees bzw Oktrees sind vor allem im Bereich der Computergrafik beliebte und oft genuzte Datenstrukturen. Für verschiedene Anwendungen, wie z.B. das Ray Tracing, wird die Berechnung eines Strahlenverlaufs durch einen Oktree benötigt. Hierfür sind vielerlei Algorithmen vorhanden. Diese Ausarbeitung teilt sich in drei Kapitel auf. Im ersten Kapitel wird unter anderem auf die Nachteile der herkömmlichen Algorithmen hingewiesen. Im zweiten Teil werden effizientere Algorithmen vorgestellt. Im letzten Teil wird ein Ray Tracing Verfahren vorgestellt, das ohne die übliche Nachbarschaftssuche auskommt.

Keywords: ray tracing, octree, quadtree

1 Einleitung

Ein Quadtree ist eine hierarchische Baumstruktur, in der jeder Knoten genau einen Vaterknoten und 4 Kinderknoten besitzt. Diese Struktur kann z.B. zur Komprimierung von Bildern, als Zugriffsmechanismus in Datenbanksystemen oder zur schnellen Berechnung beim Ray Tracing verwendet werden. Ein Quadtree kann z.B. dazu verwendet werden, eine zweidimensionale Struktur zu speichern. Jeder Knoten repräsentiert dabei eine Fläche. Ist die Fläche bezüglich einer ausgewählten Eigenschaft homogen, dann ist der Knoten ein Blattknoten ohne Kinder, ansonsten wird der Knoten in vier gleich grosse Kindknoten aufgeteilt. Dieser Vorgang wird so lange wiederholt, bis entweder eine vorgegebene Auflösung erreicht ist, oder keine neuen Kindknoten mehr generiert werden.
Im dreidimensionalen Raum werden anstatt der Quadtrees dann Oktrees (also acht statt vier Kindknoten) benützt. Um den Strahlenverlauf beim Ray Tracing zu berechnen, kann ein Oktree verwendet werden, dessen Knoten angeben, ob bzw welche Objekte im jeweilige Raumvolumen enthalten sind.
Nach dem Berechnen des Blattknotens, durch den der Strahl in den Wurzelknoten eintritt, kann der Strahlverlauf anhand der Baumstruktur verfolgt werden. Auf diese Weise kann festgestellt werden, welche Objekte ein Strahl trifft.
Häufig auftretende Probleme beim Durchlaufen des Baumes sind zum einen das Zuordnen eines gegebenen Punktes zu einem Knoten im Baum (point location), das Finden aller Knoten einer gegebenen Region (region location) und die Suche der Nachbarknoten eines gegebenen Knotens.
Da beim Ray Tracing nicht nur ein Lichtstrahl verwendet wird, sondern sehr viele, ist es wichtig, die oben genannten Probleme möglichst effizient zu lösen.
Im folgenden werden Quadtrees als Datenstrukturen verwendet. Die Übertragung der Verfahren in höhere Dimensionen lässt sich entsprechend herleiten.

2 Herkömmliche Verfahren

Herkömmliche Verfahren können in zwei Gruppen unterteilt werden. Zum einen die Gruppe der Top-Down-Verfahren, die ausgehend vom Wurzelknoten arbeiten und zum anderen die Bottom-Up-Verfahren, die vom Blattknoten in Richtung Wurzelknoten arbeiten.
Ein typisches Bottom-Up-Verfahren ist das intuitive Verfahren zur Nachbarknoten-Suche (*neighbour searches*). Hierbei wird der Baum vom Startknoten aus zum kleinsten gemeinsamen Vaterknoten (im schlechtesten Fall der Wurzelknoten) rekursiv durchlaufen. Von diesem Knoten aus wird dann im Baum abgestiegen, bis die Nachbarn des Knotens erreicht werden.
Ein herkömmliches Verfahren, einem gegebenen Punkt einen Baumknoten zuzuordnen (*point location*) ist ein typisches Top-Down-Verfahren. Im aktuellen Knoten werden die Koordinaten des Punktes mit den Koordinaten der Kinder des Knotens verglichen und dann die Suche im entsprechenden Kindknoten fortgesetzt. Dies geschieht so lange bis die unterste Ebene erreicht ist. Der Knoten, in dem die Suche endet, ist dann der gesuchte Knoten.

2.1 Nachteile dieser Verfahren

Moderne Prozessoren arbeiten meist paralell und verwenden hierfür Pipelines und Prefetch-Units, mit denen die Ausführung der Programme beschläunigt wird. Auf der Basis einer Sprungvorhersageeinheit wird im voraus entschieden, welcher Zweig des Programmes wahrscheinlich als nächstes ausgeführt werden wird. Dieser Teil wird dann von den Prefetch-Units in die Pipelines geladen und ausgeführt. Stellt sich im nachhinein (nach dem die Sprung-Entscheidung im Programm wirklich getroffen wurde) heraus, dass die vorhergesagte Entscheidung falsch war, so muss die Pipeline geleert werden und es kommt zu sogenannten Strafzyklen. Im Gegensatz zu Schleifen, bei denen die Sprungvorhersage eine hohe Trefferquote erreicht, wird beim Durchlaufen des Baumes auf Grund der schlechten Vorhersagbarkeit nur eine sehr schlechte Trefferquote erreicht, d.h. diese Art des Durchlaufens erzeugt im Normalfall viele Strafzyklen und ist damit ineffizient.
Zudem sind viele dieser Algorithmen rekursiv. Dies hat (im Vergleich zu iterativen Algorithmen) den Nachteil, dass zur normalen Rechenzeit noch die Verwaltungszeit für den Stack und die rekursiven Aufrufe verbraucht wird.

3 Ein effizienteres Verfahren

Das Verfahren welches in [TR2002-41] beschrieben wird, versucht obengenannte Nachteile beim Durchlaufen von Quadtrees bzw Okttrees zu vermeiden, indem die Knoten des Baumes entsprechend kodiert werden. Da sich die nun folgende Technik ohne weiteres in höher-dimensionierte Bäume übertragen lässt, wird der Anschaulichkeit halber im folgenden ein Quadtree als Baumstruktur verwendet.

*e-mail: mk10@informatik.uni-ulm.de

3.1 Verwendete Datenstruktur

Als Datenstruktur wird ein normaler Quadtree benutzt. Für jeden Knoten in dem Baum wird eine Verknüpfung auf den Vaterknoten (beim Wurzelknoten ist diese NULL) und auf die vier Kindknoten (hier ist die Verknüpfung ebenfalls NULL, falls es sich um einen Blattknoten handelt) gespeichert. Zudem wird zu jedem Knoten ein Zeiger auf die Daten, die der Knoten in der jeweiligen Anwendung repräsentieren soll, gespeichert.
Die Tiefe des Baumes wird auf N Ebenen festgelegt, wobei die Blattknoten die Ebene 0 bilden und die ıWurzel-Ebene die größte Ebene ist ($\equiv Ebene\_(N-1)$) Zudem werden die Werte der Knoten auf den Bereich [0..1]x[0..1] normiert.
Jedem Knoten wird ein binärer Code mit LSB-Formatierung [1], ein sogenannter *location code* zugeteilt . Dieser Code repräsentiert die Sprungentscheidungen in jedem Zweig des Baumes, d.h. der Code enthält ebensoviele Bits, wie der Baum Ebenen. In einem höherdimensionalen Baum werden die Codes für die jeweiligen Koordinatenteile unabhängig generiert.
Die Generierung des Codes für eine Zelle kann auf zwei Arten erfolgen:

- Zum einen der Code berechnet werden, indem die Koordinaten der linken oberen Ecke mit $2^{Wurzel-Ebene}$ multipliziert und das Ergebnis in Binärform dargestellt wird.
- Alternativ dazu kann auch der Weg vom Wurzelknoten zum zu bezeichnenden Knoten verfolgt und die Entscheidungen direkt als Code aufgetragen werden. (Siehe Abbildung 1)

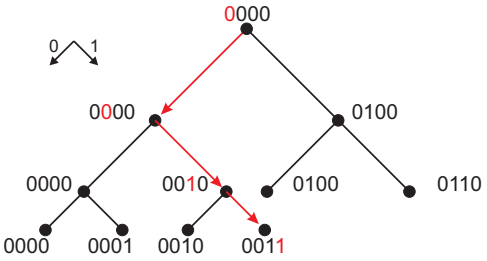

Abbildung 1: Generieren eines Knoten-Codes in einem Binärbaum

3.2 Zuordnen eines Punktes in einen Knoten (*point location*)

Ziel einer Punktsuche ist das Finden des Knotens, in dem der Punkt liegt. Im folgenden wird angenommen, dass der Wertebereich des Baumes (hier ein Quadtree) auf [0..1] x [0..1] und der des Punkt innerhalb des Bereiches [0..1) x [0..1) liege. In einem ersten Schritt werden die Koordinaten des Punktes in den *location code* umgewandelt.
Im zweiten Schritt wird ausgehend vom Wurzel-Knoten der Baum anhand der Bits bis zum entsprechenden Blattknoten durchlaufen. Hierbei dienen der generierte Code des Punktes als "Wegweiser" in dem z.B. auf der k-ten Ebene das k-te Bit des Codes als Entscheidung verwendet wird. Da die Anordnung der Kindknoten für alle Knoten gleich ist, kann der Code direkt als Index auf die Kindknoten verwendet werden.
Dieser Algorithmus benötigt zwar ebenfalls die Entscheidung, ob ein Knoten ein Blattknoten ist, jedoch fallen die zwei der restlichen beim Okttree vier) Entscheidungen weg.

[1] Least Significant Bit First

3.3 Regionen im Baum finden (*region location*)

Das Ziel dieser Methode ist es, für eine gegebene Region den kleinsten möglichen Knoten im Baum zu finden, in dem sich die Region befindet. Im folgenden wird angenommen, dass die Region rechtwinklig und paralell zu den Koordinatenasystemchsen ist.
Um den kleinsten Knoten zu finden, werden die beiden X-Werte (des rechten und linken Randes) und die beiden Y-Werte (oberer und unterer Rand) mit XOR verknüpft. Die beiden so entstandenen Codes werden nun nach dem ersten Bit (von links nach rechts) mit dem Wert 1 durchsucht. Dieses Bit gibt an, auf welcher Ebene im Baum sich der gesuchte Knoten befindet.
Bsp: $x_0 \oplus x_1 = 0010$ $y_0 \oplus y_1 = 0100 \Rightarrow$ Der erste gemeinsame Einser befindet sich an der 2. Stelle $\Rightarrow$der gesuchte Knoten befindet sich auf der Wurzel-Ebene.
In einem letzten Schritt wählt man einen beliebigen Punkt aus der Region und sucht den passenden Knoten im Baum. Hierbei wird der Baum von der Wurzel nach so lange nach unten durchlaufen, bis entweder ein Blattknoten oder die vorher berechnete Ebene erreicht wird. Wie bei der *point location* wird der Baum anhand der Bits in den XOR-Codes durchlaufen.

3.4 Nachbarknoten-Suche (*neighbour searches*)

Es gibt viele verschiedene Varianten der Nachbarknoten-Suche. Man kann entweder alle Nachbarn suchen, nur die Nachbarn in einer vorgegebenen Richtung oder von einer bestimmten Grösse.
Bei der vorliegenden Kodierung der Knoten mit *location codes* sind die Codes zweier Nachbarknoten jeweils um die Differenz des binären Abstandes der beiden Knoten verschieden.
Um beispielsweise den x-Koordinate aller rechten Nachbarn eines Knotens zu bestimmen, muss lediglich der Code der x-Koordinate mit der binären Seitenlänge des aktuellen Knotens addiert werden. Die aktuelle Seitenlänge ergibt sich jedoch direkt aus der Ebene in der der aktuelle Knoten liegt: Seitenänge $\equiv 2^{Knotenebene}$ im Binärformat
Je nach dem, welcher Nachbar gesucht wird, werden unterschiedliche Kombinationen der Koordinatenteile (x,y und gegebenenfalls z) und der Seitenlängen gebildet. Um zum Beispiel den Nachbarn rechts vom gesuchten Knoten zu finden, der grösser oder gleich dem aktuellen Knoten ist, wird der rechte Rand des Knotens berechnet (=Linker Rand + Seitenlänge). Anschliessend wird der Baum von der Wurzel an nach dem berechneten Code durchsucht. Sobald man auf der gleichen Ebene angekommen ist, oder einen Blattknoten erreicht hat, hat man den gesuchten Nachbarknoten gefunden.

In einigen Fällen ist es jedoch sinnvoll, nicht von der Wurzel aus zu starten, sondern vom aktuellen Knoten bis zum kleinsten gemeinsamen Vaterknoten von Nachbar und aktuellem Knoten auzusteigen um dann von dort aus den Baum wieder bis zum gesuchten Nachbarknoten zu durchlaufen [Samet 90b].
Die Bestimmung des kleinsten gemeinsamen Vaterknotens kann ebenfalls über die Codes laufen: Ähnlich wie beim Regionenfinden wird der Code des Knoten über XOR mit dem seines Nachbarn verknüpft um so einen Abstandscode zu generieren. Anschließend wird vom aktuellen Knoten aus der Baum so lange nach oben durchlaufen, bis die erste Binärstelle im Abstandscode Null ist. Dadurch wird automatisch der kleinste gemeinsame Vaterknoten erreicht.

Im Vergleich zu den herkömlichen Algorithmen kann dieses Verfahren ohne weiteres an die verschiedenen Varianten der Nachbarknoten-Suche und an höhere Dimensionen angepasst werden.
Außerdem benutzt diese Variante einfache Bitmanipulationen, welche direkt in den Registern der CPU ausgeführt werden können.

Viele herkömlichen Algorithmen, unter anderem [Samet 90b] stützen sich im Gegensatz dazu zudem auf Tabellen, welche, sofern sie nicht klein genug sind, nicht komplett im Cache der CPU gehalten werden können und deshalb zusätzliche Zeit beim Ein- und Auslagern benötigen.
Zudem ist die vorgestellte Methode größtenteils iterativ und benötigt weniger Vergleiche als herkömmliche Methoden.

3.5 Durchwandern des Baumes entlang eines Strahles (*ray traversal*)

Um den Vorgang des Ray Tracings zu beschleunigen, werden häufig Baumstrukturen wie der Okttree eingesetzt. Diese Baumsturkturen enthalten dann Informationen, ob bzw welche Objekte sich in welchen Knoten des Baume befinden. Der Strahl wird so lange durch die Knoten des Oktrees verfolgt, bis er auf das erste nicht-leere Blatt trifft. Auf diese Weise kann festgestellt werden, auf welches Objekt der Strahl auftrifft.
Nachdem der kleinste Eintrittsknoten festgestellt wurde (z.B. durch das Berechnen des Schnittpunktes des Wurzelknotens mit dem Strahl und anschliesender Punktsuche) kann der Strahl anhand der oben beschriebenen Nachbarknotensuche durchlaufen werden.
Eine Übersicht über die verschiedenen Algorithmen aus diesem Bereich kann unter [Havran 9] nachgelesen werden.

4 Ein alternatives ray tracing - Verfahren

Im zweiten Teil dieser Ausarbeitung soll das ray tracing - Verfahren von [X31] vorgestellt werden. Dieses Verfahren basiert auf der Parameterdarstellung des Strahles und berechnet die Parameter, in denen der Strahl die jeweiligen Knoten des Quadtrees bzw Oktrees schneidet. Im folgenden wird der Algorithmus zuerst im zweidimensionalen Raum anhand eines Quadtrees beschrieben. Anschliessend wird der Algorithmus für den dreidimensionalen Raum erweitert.

4.1 Der 2D-Algorithmus

Ein Strahl r lässt sich in parametrisierter Form als ein Paar aus Aufpunkt p ($=\binom{p_x}{p_y}$) und normiertem Richtungsvektor d ($=\binom{d_x}{d_y}$) beschreiben. Alle Komponenten des Richtungsvektor seien im folgenden ≥ 0. Alle Punkte auf dem Strahl genügen dann folgenden Gleichungen:

$$\begin{aligned} x_r(t) &= p_x + t d_x \\ y_r(t) &= p_y + t d_y \end{aligned} \tag{1}$$

Ein Knoten k (Begrenzungspunkte $\binom{x_0}{y_0}$ und $\binom{x_1}{y_1}$) im Baum wird als eine Punktemenge definiert, für die gilt

$$\begin{aligned} & x_0(k) \leq x \leq x_1(k) \\ \wedge \quad & y_0(k) \leq y \leq y_1(k) \end{aligned} \tag{2}$$

Wenn ein Strahl r nun einen Knoten k trifft, so muss folgendes gelten:

$$\begin{aligned} \exists\, t: \quad & x_0(k) \leq x_r(t) \leq x_1(k) \\ \wedge \quad & y_0(k) \leq y_r(k) \leq y_1(k) \end{aligned} \tag{3}$$

Der Eintrittspunkt $t_0(k,r)(=\binom{t_{x0}}{t_{y0}})$ und der Austrittspunkt $t_1(k,r)(=\binom{t_{x1}}{t_{y1}}$ des Strahls aus dem Knoten können wie folgt berechnet werden:

$$\begin{aligned} x_r(t_{x0}(k,r)) &= x_0(k) \quad x_r(t_{x1}(k,r)) = x_1(k) \\ y_r(t_{y0}(k,r)) &= y_0(k) \quad y_r(t_{y1}(k,r)) = y_1(k) \end{aligned} \tag{4}$$

Indem die Inverse Funktion von (1) in (4) eingesetzt wird, können die Parameter explizit definiert werden

$$\begin{aligned} t_{xi}(k,r) &= (x_i(k) - p_x)/d_x \\ t_{yi}(k,r) &= (y_i(k) - p_y)/d_y \quad i \in \{0,1\} \end{aligned} \tag{5}$$

Diese vier Werte müssen für jeden Knoten berechnet werden. Um eine iterative Berechnung zu Ermöglichen, werden die folgenden Parameter eingeführt:

$$\begin{aligned} \Delta t_x(k,r) &= t_{x1}(k,r) - t_{x0}(k,r) \\ \Delta t_y(k,r) &= t_{y1}(k,r) - t_{y0}(k,r) \end{aligned} \tag{6}$$

Durch das Einsetzen von (4) in (5) erhalten wir $\Delta t_x(k,r) = s(k)/d_x$ wobei $s(k)$ die Seitenlänge des Knotens ist. Sollte der Knoten k noch Kindknoten k_i haben, so lässt sich deren $\Delta t_x(k,r)$ und $\Delta t_y(k,r)$ schrittweise berechnen:

$$\begin{aligned} \Delta t_x(k_i,r) &= \Delta t_x(k,r)/2 \\ \Delta t_y(k_i,r) &= \Delta t_y(k,r)/2 \end{aligned} \tag{7}$$

Die Eckpunkte der Unterknoten können jeweils über folgende Rekursionsformel aus den Eckpunkten des Vaterknotens berechnet werden:

$$\begin{aligned} x_0(k_i) &= x_0(k) + s(k_i)\Delta x_i \quad \Delta x_i = \{0,0,1,1\} \\ y_0(k_i) &= y_0(k) + s(k_i)\Delta y_i \quad \Delta y_i = \{0,1,0,1\} \end{aligned} \tag{8}$$

Dadurch wird zudem eine Nummerierung der Knoten im Baum (hier im Okttree) erreicht :

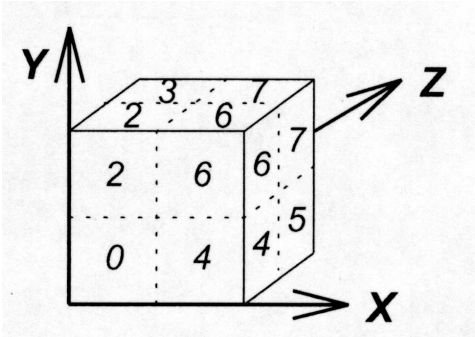

Abbildung 2: Der versteckte Knoten trägt die Nummer 1

Durch Einsetzen von (8) in (5) erhalten wir folgende Gleichung für den Eintrittspunkt (analog für $t_{y0}(k_i,r)$) mit der diese Werte schrittweise berechnet werden können:

$$t_{x0}(k_i,r) = t_{x0}(k,r) + \Delta t_x(k_i,r)\Delta x_i \tag{9}$$

Wie aus (2) zu ersehen ist, existiert genau dann ein nicht leeres Invervall der Parameter t, wenn der Knoten von dem Strahl getroffen wird. Die Bedingung (2) lässt sich nun folgendermassen umscheiben:

$$\begin{aligned} & t_{x0}(k,r) \leq t \leq t_{x1}(k) \\ \wedge \quad & t_{y0}(k,r) \leq t \leq t_{y1}(k) \end{aligned} \tag{10}$$

Wenn ein t existiert, dass die Bedingung (10) erfüllt, dann muss auch folgende Bedingung gelten

$$\begin{aligned} & t_{min} \leq t \leq t_{max} \\ \wedge \quad & t_{min}(k,r) \leq t_{max}(k,r) \end{aligned} \tag{11}$$

$$\begin{aligned} \textit{mit} \quad t_{min}(k,r) &= min(t_{x0}(k,r), t_{y0}(k,r)) \\ t_{max}(k,r) &= max(t_{x1}(k,r), t_{y1}(k,r)) \end{aligned} \tag{12}$$

Wenn Bedingung (11) erfüllt ist, dann trifft der Strahl den aktuellen Knoten. Alle Punkte des Strahles, die in dem aktuellen Knoten liegen sind im Intervall $[\, t_{min}, t_{max}\,)$ enthalten.

Um den Strahlenverlauf im Baum zu verfolgen geht man nun folgendermasen vor (Als Startknoten wird der Wurzel-Knoten verwendet.):

1. Zuerst wird die Bedingung (11) überprüft. Ist die Bedingung nicht erfüllt, so trifft der Strahl den Oktree nicht und die Rekursion endet.
2. Im anderen Fall wird der Eintritts- und der Austrittspunkt mit (5) berechnet. Ist der aktuelle Knoten ein Blattknoten, so wird die Rekursion abgebrochen und der Knoten in die Ergebnismenge eingefügt.
3. Ansonsten wird 1. für alle Kindknoten des aktuellen Knotens ausgeführt.

Bedingt durch den Aufbau des Baumes werden mehrere Werte doppelt berechnet. So ist z.B. der Eintrittspunkt in den aktuellen Knoten zugleich Eintrittspunkt in einen Kindknoten, gleiches gilt für den Austrittspunkt. Zudem ist der Austrittspunkt eines Kindknotens der Eintrittspunkt seines (sofern vorhanden) Nachbarknotens. Insgesammt gibt es je Kindknoten zusätzlich zu den Werten aus dem Vaterknotens $t_{x0}, t_{y0}, t_{x1}, t_{x1}$ noch folgende beiden Werte:

$$\begin{aligned} t_{xm}(k,r) &= \frac{(t_{x0}(k,r)+t_x(k,r))}{2} \\ t_{ym}(k,r) &= \frac{(t_{y0}(k,r)+t_y(k,r))}{2} \end{aligned} \tag{13}$$

Um den Algorithmus zu optimieren kann anstatt des Durchsuchens jedes Kindknoten des aktuellen Knotens folgender Algorithmus zur Auswahl der Kindknoten verwendet werden:

1. Suche den ersten vom Strahl getroffenen Kindknoten.
2. Solange noch nicht der Austrittspunkt des Vaterknotens erreicht ist, wählet an Hand der Werte (siehe auch Abbildung 3) den entsprechenden Nachbarknoten aus und wiederholt Schritt zwei.

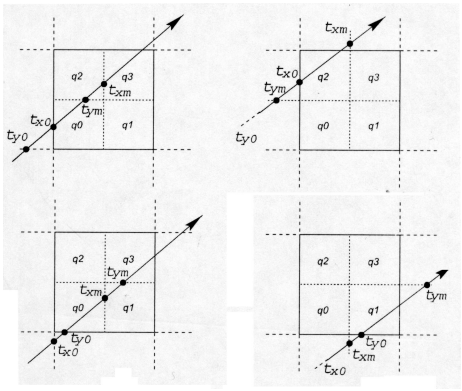

Abbildung 3:

wenn $t_{min}(k,r) < t_{max}(k,r)$ dann wird der Knoten getroffen
  wenn $t_{ym}(k) < t_{xm}(k)$ dann wird q_1 getroffen
    wenn $t_{x0}(k) < t_{<ym}(k)$ dann wird q_0 getroffen
    wenn $t_{xm}(k) < t_{<y1}(k)$ dann wird q_3 getroffen
  sonst $(t_{ym}(k) > t_{xm}(k))$dann wird q_1 getroffen
    wenn $t_{y0}(k) < t_{<xm}(k)$ dann wird q_0 getroffen
    wenn $t_{ym}(k) < t_{<y1}(k)$ dann wird q_3 getroffen
sonst wird der Knoten nicht getroffen (Bedingung (11))

4.2 Der 3D-Algorithmus

Um den 2D-Algorithmus auf einen 3D-Algorithmus zu erweitern, wird bei den einzelnen Berechnungen jeweils eine Variable für die die dritte Dimension mit einbezogen. Der Algorithmus funktioniert ähnlich dem in [Agate91] beschriebenen rekursiven HERO Algorithmus.
Der Hauptunterschied besteht im zweiten Teil des Algorithmus, vor allem darin, in welcher Reihenfolge die Kindknoten durchsucht werden. Die Bestimmung der Kindknoten entlang des Strahles in zweiten Teil geschieht über einen parametrisierten DDA-Algorithmus[2] [Amanda87]. Ein DDA-Algorithmus kann unter anderem dazu verwendet werden, einen Strahl auf ein gegebenes Raster (hier die Blattknoten) zu anzupassen (z.B. um eine gegebene Linie im Pixelraster zu zeichnen).

Um das erste vom Strahl getroffene Voxel zu bestimmen wird jedoch der Schritt 1 speziell auf den 3D-Fall angepasst, so dass dort keine Nachbarknotenssuche mehr notwendig ist.

4.2.1 Schritt 1: Berechnen des Startknotens

Um den Kindknoten zu berechnen, in dem der Strahl als erstes eintritt, wird zuerst bestimmt, in welcher Knotenseite der Strahl in den aktuellen Knoten eintritt. Dies geschieht über das Berechnen von $max(t_{x0}(k), t_{y0}(k), t_{z0}(k))$. Die Werte für $t_{z0}(k)$ können analog zu (9) wie die $t_{x0}(k)$-Werte im 2D-Fall berechnet werden.
Anhand von Tabelle 1 kann dann die Eintrittsseite festgestellt werden, wodurch nur noch vier mögliche Kindknoten als Kandidaten übrigbleiben.
Nach dem Berechnen der Werte $(t_{xm}(k), t_{ym}(k), t_{zm}(k))$ können die

Maximum	Entry plane
t_{x0}	YZ
t_{y0}	XZ
t_{z0}	XY

Abbildung 4: Tabelle 1

in Tabelle 2 beschriebenen Vergleiche berechnet werden. Indem je nach Eintrittsseite die entsprechenden Vergleiche ausgeführt, und deren Ergebnisse mittels binärem OR an der entsprechenden Stelle verknüpft wurden, erhält man die Nummer des Eintrittsknotens (siehe auch Abbildung 2) in binärer Darstellung.
Durch die in der Tabelle beschriebenen Vergleiche können alle möglichen Kindknoten außer dem Knoten mit der Nummer sieben erreicht werden. Dieser Knoten kann mit den obrigen Einschränkungen (die Komponenten des Richtungsvektors müssen größer als Null sein) von jedoch auch keinem eintreffenden Strahl erreicht werden.

Entry Plane	Conditions to examine	Bit affected
XY	$t_{xm}(o) < t_{z0}(o)$	0
	$t_{ym}(o) < t_{z0}(o)$	1
XZ	$t_{xm}(o) < t_{y0}(o)$	0
	$t_{zm}(o) < t_{y0}(o)$	2
YZ	$t_{ym}(o) < t_{x0}(o)$	1
	$t_{zm}(o) < t_{x0}(o)$	2

Abbildung 5: Tabelle 2

[2] Digital Differential Analyser

4.2.2 Schritt 2: Auswahl des nächsten Knotes

Nachdem nun der erste Eintrittsknoten feststeht und die Kindknoten innerhalb eines Vaterknotens immer gleich angeordnet sind, kann die Auswahl des nächsten Knotens, durch einen einfachen Zustandsautomat erfolgen.
Sei der aktuelle Knoten k_i. Um den Strahlenverlauf zu berechnen ist es notwendig, die Austrittsseite des Strahles zu ermitteln. Dies geschieht über Berechnen von $min(t_{x1}(k_i), t_{y1}(k_i), t_{z1}(k_i))$. Anhand dieser Werte und der Tabelle 3 kann der Nachfolgerknoten vom Zustandsautomat bestimmt werden. In Tabelle 3 wird der zur Auswahl notwendige Zustandsautomat definiert, wobei "End" bedeutet, dass der Strahl den Vaterknoten k verlässt und die Suche in diesem Knoten abgeschlossen ist.

Current sub-node (state)	Exit plane YZ	Exit plane XZ	Exit plane XY
0	4	2	1
1	5	3	End
2	6	End	3
3	7	End	End
4	End	6	5
5	End	7	End
6	End	End	7
7	End	End	End

Abbildung 6: Tabelle 3

Ist beispielsweise der Eintrittsknoten der Knoten 3, so kann nur noch der Knoten 7 als einziger Nachfolger in Frage kommen, da alle Komponenten des Richtungsvektors des Strahles größer als Null sein müssen.
Im Gegensatz zum HERO-Algorithmus, der alle möglichen Kindknoten nacheinander darauf prüft, ob sie vom Strahl getroffen werden, kann dieser Algorithmus direkt die entsprechenden Knoten auswählen und vermeidet somit die Berechnung der nicht getroffenen Knoten.

4.2.3 Anpassen des Algorithmus an beliebige Strahlenverläufe

Der bisher vorgestellte Algorithmus ist nur für Strahlen ausgelegt, deren Richtungsvektorkomponenten alle grösser als Null sind. Strahlen, die paralell zu den Koordinatenachsen oder eine negativer Richtungskomponente enthalten können mit den folgenden Modifikationen ebenfalls vom Algorithmus behandelt werden.

Im folgenden wird festgelegt, dass Strahlen, die paralell zu einer Ebene verlaufen, diese im Unendlichen treffen. Dadurch wird der Wertebereich der Parameter und Variablen formal auf $[-\infty..+\infty]$ aufgedehnt. Um die Berechnungen aus (1) - (13) im neuen Wertebereich ausführen zu können, werden folgende Definitionen eingeführt:

$$\begin{array}{ll} x/0 = +\infty & \forall\, x > 0 \\ x/0 = -\infty & \forall\, x < 0 \end{array} \tag{14}$$

Für den Fall, dass $d_x = 0$ kann der Strahl einen Knoten nur treffen, wenn $t_{x0} = -\infty$ und $t_{x1} = +\infty$ Zur Berechnung von t_{xm} und t_{ym}, siehe (13) wird dann folgende Definition benötigt (Analoges gilt für die annderen Parameter):

$$t_{xm}(k,r) = \begin{cases} +\infty & wenn p_x < \frac{x_0(k)+x_1(k)}{2} \\ -\infty & sonst \end{cases} \tag{15}$$

Durch diese Definitionen wird sichergestellt, das die entsprechenden Vergleiche (z.b. in Tabelle 3) die richtigen Ergebnisse liefern.

Um Strahlen mit negativen Richtungskomponenten mit dem Algorithmus zu berechnen, werden diese zuerst an den Mittelachsen des Wurzelknotens gespiegelt. Auf diese Weise werden die negativen Komponenten in positive Komponenten umgewandelt:

$$\begin{array}{c} d'_x = -d_x \\ p'_x = s(k) - p_x \end{array} \tag{16}$$

Zu jeder der möglichen Achsspiegelungen wird eine Funktion f benötigt, die dann die vom Strahl durchlaufenen Knoten entsprechend umnummeriert. In Abbildung 7 wird der Strahl r mit negativer x-Richtungskomponente, der den Knoten auf dem Weg (4,6,2) durchläuft, an der Mittelachse des Knotens gespiegelt. Dadurch entsteht der Strahl r'.
Dieser durchkreuzt den Knoten jedoch auf dem Weg (0,2,6). Es wird also eine Funktion benötigt, die dafür sorgt, wenn der Zustandsautomat den Knoten i als nächsten Knoten berechnet, statt dessen der richtige Knoten f(i) ausgewählt wird.

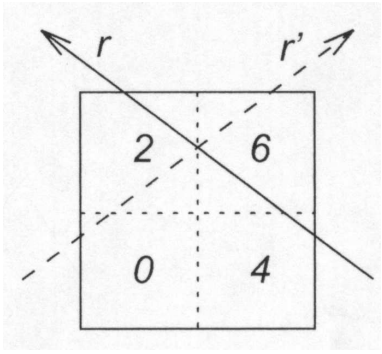

Abbildung 7: Spiegelung des Strahles r mit negativer x-Komponente an der Mittelachse

In diesem Beispiel würde der Bildbereich der Funktion f (4,5,6,7,0,1,2,3) sein, d.h. die Funktion ändert jeweils das höchstwertige Bit der Binärdarstellung der Knotenmummer.
Generell müssen bei jeder Spiegelung des Strahls die Knotennummern geändert werden. Dies kann dadurch erreicht werden,für eine negative Komponente dass jeweils der Komponente zugeordnete Bit in der Knotennummer umgeklappt wird. Folgender Funktion realisiert dies:

$$\begin{array}{c} f(i) = i \oplus a \\ mit \quad a = 4a_x + 2a_y + a_z \end{array} \tag{17}$$

$$wobei \begin{cases} a_e = 1 & falls\ an\ der\ e-Achse\ gespiegelt\ wurde \\ a_e = 0 & sonst \end{cases} \tag{18}$$

4.2.4 Bewertung des Algorithmus

Um den Algorithmus mit verschiedenen anderen Algorithmen zu vergleichen, verwenden die Authoren [X31] verschiedene Eingabeszenen. Zum einen eine Wolke aus 1890 Sphären (Siehe Anhang,

Abbildung 11 ,Ergebnis in Abbildung 8), eine aus 4320 Zylindern und Sphären (Siehe Anhang, Abbildung 12, Ergebnis in Abbildung 9) und eine aus 4350 Objekten (Siehe Anhang, Abbildung 13, Ergebnis in Abbildung 10) und bestehende Szene. Diese Szenen wurden in Oktress der Tiefe 5, 6, 7 und 8 umgewandelt.
Als Vergleichskandidaten wurden die Top-Down-Algorithmen Samet [Samet89]*(Sam)*, SametCorner [Endl94]*(SamC)* und SametNet [Endl94] *(SamNet)* hinzugezogen. Gargantini [Garga93] *(Gar)* wurde als Vertreter der Bottom-Up-Methoden verwendet.
Wie aus den Grafiken der Autoren ersichtlich ist, ist der von Autoren von [X31] vorgestellten Algorithmus *(Par)* etwa gleich schnell wie der *Gar*, in den meisten Fällen ist er sogar etwas effizienter.

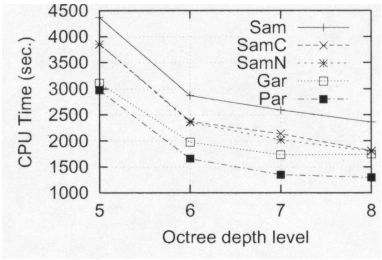

Abbildung 8: Vergleichstabelle unter Verwendung einer Wolken-Szene mit 1890 Sphären

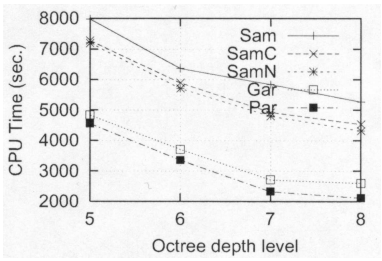

Abbildung 9: Vergleichstabelle unter Verwendung einer Szene aus 4320 Zylindern und Sphären

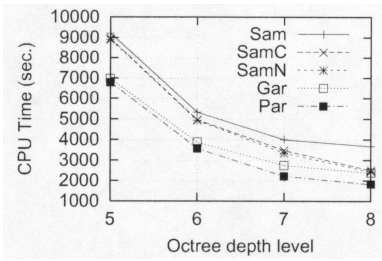

Abbildung 10: Vergleichstabelle unter Verwendung einer Szene aus 4350 Objekten

Desweiteren vergleichen die Autoren den Algorithmus mit dem in [Sung92] vorgestellten Algorithmus, der anstatt eines Oktrees einen Binärbaum verwendet. Als Eingabeszene wurde hier ein dreidimensionales Sphärengitter (Siehe Anhang, Abbildung 14) aus 2^{3n} gewählt. Es wurden drei Durchläufe ausgeführt, mit n=3 ($\rightarrow$ 512 Sphären), n=4 ($\rightarrow$ 4096 Sphären) und n=5 ($\rightarrow$ 32768 Sphären). Bei diesen Durchläufen stellten die Autoren fest, dass der Oktree-Algorithmus im Vergleich zum Algorithmus aus [Sung92] um 10.4%, 12.7% bzw 6.8% schneller war.
Da beide Algorithmen bis auf die verwendete Datenstruktur annähernd gleich arbeiten (siehe [ReinH96]),erklären die das Ergebnis Autoren dadurch, dass bei einem (annähernd) vollständigen Baum in der Oktree-Darstellung weniger Verzweigungen und damit auch weniger Entscheidungen und rekursive Funktionsaufrufe als bei einem Binärbaum benötigt werden.

5 Zusammenfassung

Sofern die Suche des Strahlverlaufes im Oktree schneller verläuft, als die Überprüfung jedes einzelnen in der Szene vorhandenen Objektes, eignet sich die Datenstruktur des Oktrees bzw des Quadtrees besonders um das Ray Tracing zu beschleunigen.
Zum einen lassen durch eine geschickte Kodierung der Knotennummern generell effiziente Algorithmen zum Durchsuchen und Durchlaufen des Baumes finden, zum anderen können kostenintesive Teile beim Ray Tracing, wie zum Beispiel die Nachbarknotensuche vermieden werden. Zudem können die Algorithmen dadurch optimiert werden, indem die aktuelle Hardware-Architektur der Prozessoren berücksichtigt wird (siehe 3.4)

Literatur

[TR2002-41] Sahra F. Frisken; Ronald N. Perry **Simple and Efficient Traversal Methods for Quadtrees and Octrees** Mitshubish Electric Research Laboratories

[X31] J. Revelles; C, Ureña; M. Kastra **A Efficient Parametric Algorithm for Octree Traversal**

[Samet 90b] H. Samet. **Application of Spatial Data Structures: Computer Grafics Processing** , GIS. Addison-Wesley, Reading, MA, 1990

[Havran 99] V. Havran 1990b, **A Summary of Octree Ray Traversal Algorithms**, Ray Tracing News, 12(2) pp. 11-23, 1999

[Agate91] M. Agate, R.L. Grimsdale, P.F.Lister **The HERO Algorithm for Ray Tracing Oktrees. Advances in Computer Grafics Hardware IV**, Springer-Verlag, New York, 1991

[Amana87] J. Amantides, A. Woo, **A Fast Voxel Traversal Algorithm for Ray Tracing**, Eurograhics '87, Proceedings of the European Computer Grafics Conference and Exhibition, August 1987, pp 3-10

[Sung92] K. Sung, P. Shirley, bf Ray Tracing with de BSP tree, Graphics Gems III, pp 271-274, 1992

[Samet89] H. Samet **Implementing Ray Tracing with Octrees and Neighbour Finding** Computer and Grafics Vol 13(4), pp 445-460, 1989

[Garga93] I. Gragantini, H. H. Atkinson **Ray Tracing an Octree: Numerical Evaluation of the First Intersection** Computer Graphics Forum Vol. 12(4), 1993

[Endl93] R. Endl, M. Sommer **Classification of Raygenerators in Uniform Subdivisions an Octree for Ray Tracing** Computer Graphics Forum Vol. 13(1), 1994

Anhang

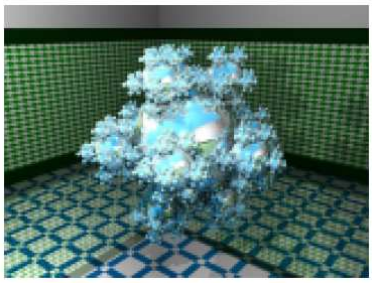

Abbildung 11: Vergleichsszene aus 1890 Sphären

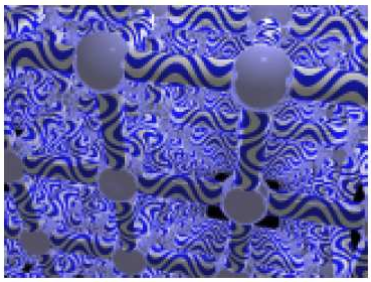

Abbildung 12: Vergleichsszene aus 4320 Zylindern und Sphären

Abbildung 13: Vergleichszene aus 4350 Objekten

Abbildung 14: Dreidimensionales Sphärengitter