Seminar Computergrafik

– Universität Göttingen –

Schriftliche Ausarbeitung zum Thema

Die Verwendung vorausberechneter Dreiecks-Cluster für beschleunigtes Raytracing in dynamischen Szenen

---

Simon Schütz
Fakultät für Physik
Universität Göttingen
simon.schuetz@stud.uni-goettingen.de

Inhaltsverzeichnis

1 Einleitung

Raytracing beschreibt in der Computergrafik eine Rendertechnik, die durch die Verfolgung von Strahlen eine durch eine Vielzahl von Primitiven (meist Polygonen) und einen Satz an Texturen und Beleuchtungsregeln beschriebene Szene auf ein zweidimensionales Rasterbild abbildet. Durch Fortschritte in der Rechenleistung moderner Computersysteme, aber auch den verwendeten Algorithmen ist es heute möglich, nicht nur statische Bilder, sondern auch interaktive Szenen und sogar Echtzeitanimationen in hinreichender Auflösung darzustellen[7]. Durch weitere Forschung sollen nun Algorithmen und Datenstrukturen entwickelt werden, die es erlauben, Echtzeitanimationen hoher Komplexität auch auf weniger leistungsstarken Computersystemen durchzuführen und somit für Heimanwender interessant zu machen.

Diese Ausarbeitung stellt den Artikel *The Use of Precomputed Triangle Clusters for Accelerated Ray Tracing in Dynamic Scenes* von K. Garanzha[1] vor und ordnet ihn in sein Umfeld ein. Dazu werden zunächst die Grundlagen des Raytracing in dynamischen Szenen mit den dazu nötigen Datenstrukturen erläutert, um dann das Konzept des neuen Beschleunigungsalgorithmus zu erklären und mit bestehenden Ansätzen zu vergleichen.

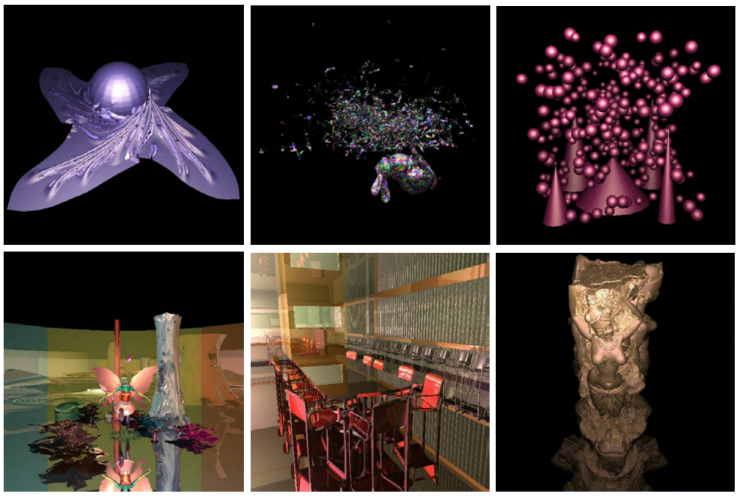

Abbildung 1: Szenen, die vom Algorithmus gerendert werden. Die ersten vier Bilder zeigen Ausschnitte von Animationen, das fünfte Bild ist Beispiel einer Szene mit sehr unregelmäßigen Dreiecken, das sechste Bild zeigt eine Szene mit sehr vielen Dreiecken. (Quelle: [1])

2 Raytracing in dynamischen Szenen

Algorithmen zur effizienten Darstellung von Szenen basieren auf Datenstrukturen, die die Geometrie der Szene schnell zugänglich machen. Statische Szenen zeichnen sich dadurch aus, dass sich die Geometrie, also die Position der Primitiven im Raum, nicht verändert. Lediglich die Position der Kamera im Raum sowie die Art der Beleuchtung kann sich mit der Zeit ändern. Dafür wird die Geometrie vor Beginn des Rendervorgangs in spezialisierten Datenstrukturen erfasst, die dann nach einer kurzen Ladezeit in Echtzeit für die gegebenen Beleuchtungsparameter und Kamerapositionen ausgewertet werden. Da der Aufwand zum Erstellen der Geometrie nur einmal vor Beginn des Anzeigevorgangs betrieben wird, sind die weiteren Schritte hinreichend günstig, dass sie für entsprechend einfache Szenen auf Computersystemen von Heimanwendern mit akzeptabler Bildrate und Auflösung dargestellt werden können[4].

2.1 Schwierigkeiten animierter Szenen

Gegenüber derart statischen Szenen sind dynamische Szenen, in denen sich ein Teil der Primitiven im Raum bewegt, schwieriger umzusetzen. Hier muss die Datenstruktur, die die Szene zugänglich macht, beständig aktualisiert werden, um die Korrektheit der Darstellung zu gewährleisten. Dicht an der Vorgehensweise für statische Szenen angesiedelt sind Fälle, in denen nur ein sehr geringer Teil der Szene sich zeitlich verändert, oder aber der zeitliche Ablauf der Animation im Voraus komplett bekannt ist und somit auch von vornherein in die Datenstruktur integriert werden kann (vgl. z.B. [4]). Darüber hinaus kann unterschieden werden[7] zwischen Animationen, in denen ganze Gruppen von Primitiven gleichartig transformiert werden, also ein rigides Objekt bilden; Animationen, in denen die Oberfläche eines Objekts, dargestellt durch ein Dreiecksnetz (Mesh), zusammenhängend bleibt, aber die Vertizes unterschiedlich transformiert werden; und Animationen, in denen sich die Topologie vollkommen ändert, etwa indem die Konnektivität des Netzes verändert oder Primitiven hinzugefügt bzw. entfernt werden. Das in Abschnitt 4 vorgestellte Verfahren[1] ist auf Anwendungsfälle zugeschnitten, in denen eine dynamische Szene durch Dreiecksnetze dargestellt wird, deren zeitliche Transformation zwar vor Beginn der Animation nicht bekannt sein muss, deren Topologie, also Konnektivität der Netze, aber für die Animation konstant bleibt.

3 Räumlich sortierende Datenstrukturen

Der elementare Schritt beim Raytracing ist, einen Strahl, der in der Szene von einem gegebenen Punkt ausgeht, so lange zu verfolgen, bis er auf ein Mesh trifft. Dazu muss in der Szene unter den Dreiecken, die der Strahl schneidet, dasjenige bestimmt werden, das dem Ausgangspunkt des Strahls am nächsten ist. Dieser Schritt wird beim Rendern sehr häufig durchgeführt, sodass die Performance des Raytracing-Algorithmus signifikant vom Aufwand für diesen Schritt abhängt. Anstatt für jedes Dreieck der Szene zu bestimmen, ob es einen Schnittpunkt mit dem gegebenen Strahl hat, und dann unter allen geschnittenen Dreiecken dasjenige auszuwählen, das dem Strahlursprung am nächsten liegt, bietet es sich an, die Meshes derart in einer Datenstruktur zu speichern, dass sie räumlich geordnet sind. In einem räumlich geordneten Suchbaum können Dreiecke, die abseits des Strahlweges liegen, von vornherein aus der Suche ausgeschlossen werden. Beginnt man nun unter denjenigen Dreiecken, die dicht am Strahl liegen, nahe an dessen Ursprung mit der Suche nach einem Schnittpunkt und findet diesen, kann man alle weiter entfernten Dreiecke ebenfalls von der Suche ausschließen und gelangt somit schnell zu einem korrekten Ergebnis.

3.1 kd-Baum

Ein einfacher in der Computergrafik verwendeter Ansatz zur Sortierung von Objekten im Raum ist der *kd-Baum*[7], wobei k die Dimension des Raumes angibt. Ein kd-Baum ist ein binärer Baum, der ein räumliches Volumen pro Tiefenebene entlang einer Koordinatenrichtung in zwei Teile teilt. Die Blätter repräsentieren somit Teilvolumina des Raumes und enthalten die im entsprechenden Volumen befindlichen Dreiecke. Dreiecke, die von einer Trennungsebene durchschnitten werden, werden dazu in Teilpolygone aufgeteilt. Durch geeignete Traversierung des Baumes werden nur diejenigen Blätter, deren Volumina der Strahl durchquert, nacheinander ausgewertet. Da die Wahl der Positionen der Trennungsebenen in Abhängigkeit der abzulegenden Dreiecke die Leistung der Struktur bestimmt, konzentrieren sich Algorithmen darauf, mittels Heuristiken und auch zusätzlichen Informationen, etwa des Szenegraphen der abzubildenden Szene, gute Werte vorzugeben. Beliebt ist hier die Oberflächenheuristik (Surface Area Heuristic, SAH), die in Abschnitt 3.3 erläutert wird.

3.2 Bounding Volume Hierarchy

Eine weitere Datenstruktur, die eine räumliche Sortierung umsetzt, ist die *Bounding Volume Hierarchy* (BVH). Dreiecke in einem Mesh werden mit ihrer *Bounding Box* in den Blättern dieses Suchbaumes abgelegt. Die Bounding Box ist dabei der-

jenige entlang der Koordinatenachsen ausgerichtete Quader, der alle Dreiecksvertizes bei minimalem Volumen umschließt[1]. Ein innerer Knoten hält jeweils diejenige Bounding Box, die die Bounding Boxen aller seiner Kinder bei minimalem Volumen umschließt. Eine derart aufgebaute räumliche Baumstruktur garantiert nun aber noch keine hohe Performance beim Raytracing; erst durch einen Algorithmus, der die Dreiecke im Baum sinnvoll einordnet, kann eine hohe Leistung erreicht werden.

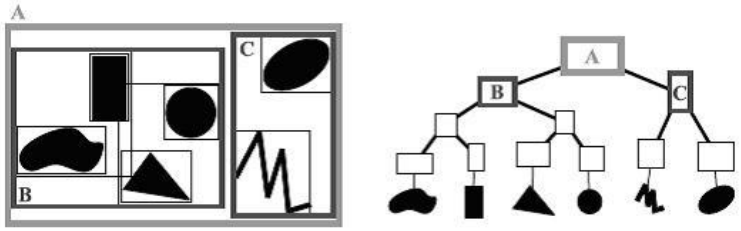

Abbildung 2: Aufbau einer Bounding Volume Hierarchy, hier in 2 Dimensionen, aber mit komplexeren Primitiven (Quelle: [5]).

Der Geschwindigkeitsvorteil bei der Verwendung von BVHs basiert darauf, dass die Bounding Boxes der Geschwisterknoten – insbesondere in geringer Tiefe – möglichst disjunkt sind, sodass ein Strahl durch die Szene eine möglichst geringe Anzahl an Bounding Boxes schneidet, die dann ausgewertet werden müssen. Der Aufbau eines guten Suchbaumes ist aber aufwendig, sodass ein Kompromiss zwischen hoher Qualität des Baumes, also geringem Aufwand für die Auswertung der Datenstruktur, und schnellem Aufbau, verbunden mit einer weniger effizienten Auswertung, gefunden werden muss. Bewährt haben sich Top-Down-Ansätze (z.B. [6]). Hier wird die Menge der Dreiecke rekursiv in Teilmengen unterteilt, deren Bounding Box dann jeweils einen inneren Knoten bildet, bis jede Teilmenge nur noch ein Element enthält, das ein Blatt wird. Wie die Unterteilung verläuft, wird wieder anhand einer Heuristik entschieden, etwa der Oberflächenheuristik. Es existieren Algorithmen, mit deren Hilfe eine BVH in animierten Szenen nicht mit jedem Frame neu aufgebaut werden muss, sondern in Teilen aktualisiert und weiter verwendet werden kann.

3.3 Oberflächenheuristik

Die *Oberflächenheuristik* (Surface Area Heuristic, SAH) schätzt die Kosten für die Traversierung eines räumlichen Suchbaumes beim Raytracing ab. Nach Wald[7] ergibt sich unter der Annahme einer homogenen Orts- und Richtungsverteilung für

[1] Im allgemeinen Fall werden Bounding Volumes betrachtet, für die lediglich gefordert wird, dass sie ein beliebiges geschlossenes Volumen sind, die das Objekt umschließen; eine an den Koordinatenachsen ausgerichtete Quaderform vereinfacht viele Operationen aber stark.

Strahlen, dass die Wahrscheinlichkeit $P_{hit}(X)$, dass ein Strahl ein konvexes Volumen X trifft, proportional mit dessen Oberfläche $SA(X)$ anwächst. Insbesondere gilt für ein Teilvolumen Y mit $Y \subset X$, wenn bekannt ist, dass der Strahl X trifft:

$$P_{hit}(Y|X) = \frac{SA(Y)}{SA(X)} \tag{1}$$

Daraus lassen sich für eine gegebene Schachtelung von Raumvolumen mit bekannter Anzahl darin enthaltener Dreiecke (also einen Knoten eines kd- oder BVH-Baums beim Top-Down-Aufbau) die erwarteten mittleren Kosten C_R abschätzen, die sich für einen Strahl R ergeben, der das Volumen durchquert: Seien nun Y_1 und Y_2 Teilvolumina von X, sei $n_i = n(Y_i)$ die Anzahl der Dreiecke im Volumen Y_i und $n(X \setminus (Y_1 \cup Y_2)) = 0$. Seien C_T die Kosten eines Traversionsschrittes und C_I die Kosten für einen Kollisionstest des Strahls mit einem Dreieck. Dann gilt für C_R näherungsweise:

$$\begin{aligned} C_R &= C_T + C_I \cdot (n(Y_1)P_{hit}(Y_1|X) + n(Y_2)P_{hit}(Y_2|X)) \\ &= C_T + C_I \cdot \frac{n_1 \cdot SA(Y_1) + n_2 \cdot SA(Y_2)}{SA(X)} \end{aligned} \tag{2}$$

Die Heuristik nimmt nun an, dass diejenige Schachtelung (Y_1, Y_2), die diese Kosten minimiert, den für das Raytracing am besten geeigneten Baum erzeugt.

4 Verwendung vorausberechneter Dreieckscluster

Der von K. Garanzha[1] entwickelte Ansatz sieht vor, den Aufbau der Bounding Volume Hierarchy zu beschleunigen, indem statt einzelner Dreiecke ganze Cluster von Dreiecken, also Meshes, übergeben werden. Das reduziert die Länge der Eingabe an den Algorithmus, der die BVH aufbaut. Da die Komplexität eines BVH-Algorithmus auf einer Eingabe der Länge n mindestens $\mathcal{O}(n \log n)$ beträgt, kann so der Aufwand pro Frame deutlich gesenkt werden. Die Kosten für die Erstellung der Cluster fallen hingegen nur einmal am Anfang an, wenn man voraussetzt, dass die Konnektivität der Meshes über die Animation konstant bleibt und keine Meshes hinzukommen oder entfernt werden, sich die Vertexindizierung über den gesamten Verlauf der Animation also nicht ändert. Für die Erzeugung der Cluster wird ein Algorithmus vorgestellt, der dicht gepackte Cluster mit entsprechend engen Bounding Boxen findet. Diese führen entsprechend der Oberflächenheuristik zu besonders effizienten BVH-Bäumen.

4.1 Clusterheuristik

Die Clusterheuristik nimmt eine Vorsortierung der Dreiecke vor, die im Standardverfahren erst beim Aufbau des BVH-Baums sortiert werden würden. Daher überrascht es nicht, dass die Heuristik, nach der der Clusteralgorithmus vorgeht, an die Oberflächenheuristik angelehnt ist. Analog zu Abschnitt 3.3 seien X und Y Raumvolumen mit $Y \subset X$, wobei nun Y ein Dreieckscluster[2] und X dessen Bounding Box ist. Die Clusterheuristik versucht nun, $P_{hit}(Y|X)$ zu maximieren: Erst dann, wenn der Strahl ein Dreieck in der Box trifft, kann die Suche beendet werden. Während die Auswertung des BVH-Baums in den inneren Knoten davon profitiert, wenn ein Strahl möglichst wenige Kinder schneidet, ist es beim Test der Blätter hilfreich, wenn es möglichst schnell zu einer Kollision kommt.

Einfluss der geometrischen Form

Die Blätter beim Standardverfahren enthalten Dreiecke. Für ein Dreieck in seiner Bounding Box ist[3]:

$$P_{hit}(Y|X) \in (0, 0.5]$$

[2] Als zweidimensionales Objekt im dreidimensionalen Raum hat das Mesh kein Volumen im physikalischen Sinn; Y ist hier als Menge von Punkten zu verstehen. Die Funktion $P_{hit}(Y)$ kann nach wie vor angegeben werden.

[3] Die Wahrscheinlichkeit verschwindet, wenn der Strahl es von der Seite trifft, und wird maximal, wenn es in einer Koordinatenebene senkrecht zum Strahl liegt und eine Seite parallel zu einer der Koordinatenachsen ist.

Für eine Kugel hingegen gilt:

$$P_{hit}(Y|X) = \frac{SA(Y)}{SA(X)} = \frac{4\pi R^2}{6 \cdot (2R)^2} = \pi/6$$

Eine kreisrunde Form hat eine Trefferwahrscheinlichkeit von[4]:

$$P_{hit}(Y|X) = \frac{SA(Y)}{SA(X)} \geq \frac{2 \cdot \pi R^2}{6 \cdot (2R)^2} = \pi/12$$

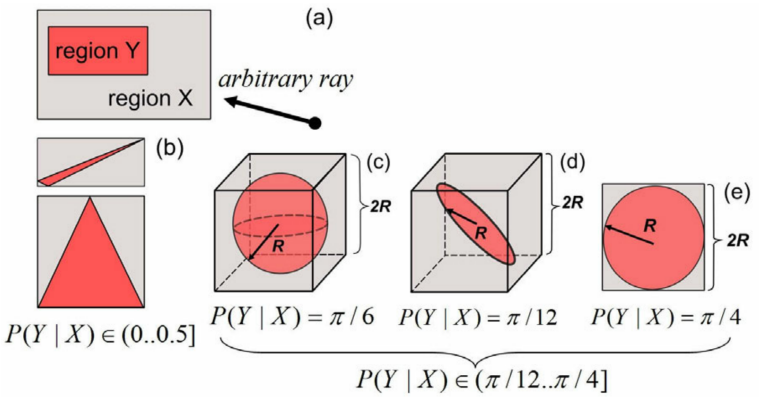

Abbildung 3: Berechnung von $P_{hit}(Y|X)$ für verschiedene Formen (Quelle: [1]).

Bei beliebiger Orientierung der Cluster ist festzustellen, dass kreisrunde oder sphärische Cluster eine höhere Trefferwahrscheinlichkeit haben als Dreiecke (vgl. Abb. 3). Durch passendes Clustern kann also nicht nur der Aufbau des BVH-Baums, sondern gleichzeitig auch seine Auswertung beschleunigt werden.

Einfluss der Konnektivität

Eine Annahme des Ansatzes für animierte Szenen ist, dass sich die zusammenhängenden Cluster im Verlauf der Animation nicht zu stark verformen. Dies gilt in der Praxis eher für Vertizes, die durch wenige Kanten verbunden sind, als für Vertizes mit geringem räumlichen Abstand, zwischen denen aber viele Kanten liegen: Cluster mit *dichter Konnektivität* verformen sich weniger. Die Konnektivität steigt mit der Anzahl der Dreiecke pro Vertex, für ein Cluster mit N_V Vertizes und N_T Dreiecken kann die Konnektivität also durch N_T/N_V beschrieben werden.

[4]In diesem Fall wird ein Quader der Kantenlänge $2R$ je nach Lage der Fläche nicht ausgefüllt, sodass P_{hit} nur abgeschätzt wird.

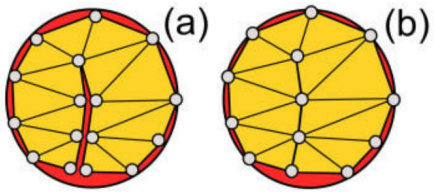

Abbildung 4: Konnektivität: Cluster (a) hat eine Konnektivität von $14/16 \approx 0.88$, Cluster (b) eine Konnektivität von $14/13 \approx 1.08$. Cluster vom Typ (b) sollten bevorzugt gebildet werden, da eher zu erwarten ist, dass die Form über die Animation ähnlich bleibt. (Quelle: [1])

4.2 Bestimmung der Zielfunktion

Die Kriterien an einen gut geeigneten Cluster werden nun formalisiert in einer Zielfunktion $Acc(k)$ zusammengefasst, wobei k für ein Cluster von k verbundenen Dreiecken steht und ein solches Cluster so gefunden werden soll, dass $Acc(k)$ möglichst groß ist.

Anpassung der geometrischen Form

Sei die *Bounding Sphere* $S(k)$ mit Radius $R(k)$ die minimale Kugel, die alle k Dreiecke umschließt. Die Fläche eines $S(k)$ einbeschriebenen Kreises ist:

$$AreaC(k) \;=\; \pi \cdot R(k)^2 \tag{3}$$

Sei $\vec{n}_i$ der normalisierte Normalenvektor des i-ten Dreiecks. Definiere den mittleren Normalenvektor $\vec{N}_{avg}$:

$$\vec{N}_{avg}(k) \;=\; \frac{\sum_{i=1}^{k} \vec{n}_i}{\left|\sum_{i=1}^{k} \vec{n}_i\right|} \tag{4}$$

Im Fall $\sum_{i=1}^{k} \vec{n}_i = \vec{0}$ ist $\vec{N}_{avg}(k)$ nicht definiert; in diesem Fall kann der Vektor beliebig gesetzt werden. Definiere nun mit der Oberfläche des i-ten Dreiecks, $Area_i$:

$$AreaP(k) = \sum_{i=0}^{k} Area_i \left|\vec{N}_{avg}(k) \cdot \vec{n}_i\right| \tag{5}$$

Diese Operation projiziert die Fläche der Dreiecke in die Ebene, die der mittleren Ausrichtung der Dreiecke entspricht. Je größer diese Fläche gegenüber der Querschnittsfläche der Kugel $S(k)$ ist, desto mehr entspricht die Form des Clusters

einer Scheibe oder Kugel. Formell sei die Regularität $reg(k)$ definiert als:

$$reg(k) \;=\; \frac{AreaP(k)}{AreaC(k)} \tag{6}$$

Die Funktion $reg(k)$ nimmt für flache Meshes Werte im Intervall $(0, 1)$ an und kann für gewölbte Oberflächen auch Werte größer als 1 erreichen.

Größenbeschränkung der Cluster

Werden die Cluster zu groß, steigt der Aufwand für den Kollisionstest pro Cluster an, während die BVH-Struktur an Effizienz verliert. Daher muss die Heuristik die Größe der Cluster einerseits in der Anzahl ihrer Dreiecke, andererseits in ihrer geometrischen Oberfläche, nach oben beschränken. Eine zu große Anzahl an Dreiecken steigert die bei der Auswertung eines Clusters anfallenden Kosten[5], eine große Oberfläche führt zu einer großen Bounding Box, die nach Oberflächenheuristik nicht erstrebenswert ist.

An Stelle einer Beschränkung für die Zahl der Dreiecke wird die Zahl der Vertizes beschränkt. Dies hat den Effekt, dass Cluster dichter Konnektivität, die relativ zur Zahl der Dreiecke weniger Vertizes haben, von der Heuristik bevorzugt werden. Sei $MaxCount$ die Zahl der erwünschten Vertizes in der Struktur. $MaxCount$ ist ein freier Parameter des Modells, der für eine spezifische Implementierung des Ansatzes so gewählt werden muss, dass er die Leistung des Algorithmus maximiert. Sei $VertexCount_k$ die Zahl der Vertizes im Cluster k. Dann gilt für die Funktion $regBound(k)$:

$$regBound(k) \;=\; \frac{VertexCount_k}{MaxCount} \tag{7}$$

Je höher der Wert von $regBound(k)$, desto mehr nähert sich k der Größenbeschränkung; im Intervall $(0, 1)$ ist diese noch nicht erreicht.

Um die Größe der Bounding Box zu beschränken, wird eine obere Schranke für die Oberfläche der Bounding Sphere $S(k)$ auf ein Vielfaches der mittleren Oberfläche der Bounding Spheres der einzelnen Dreiecke der Szene, $AvgSa$, beschränkt. Mit dem freien Parameter $MaxSize$, der wieder im Test bestimmt werden muss, gilt dann:

$$sizeBound(k) \;=\; \max(1, \frac{SA(S(k))}{MaxSize \cdot AvgSA}) \tag{8}$$

[5]Während der Aufwand zu Auswertung eines Knotens des BVH-Baums logarithmisch in der Anzahl seiner Knoten ansteigt, ist der Aufwand innerhalb eines Clusters linear zur Zahl seiner Dreiecke.

Der Wert von $sizeBound(k)$ liegt im Intervall $(0, 1]$, wobei ein kleinerer Wert für ein Cluster erstrebenswert ist. Das Produkt $regBound(k) \cdot sizeBound(k)$ kombiniert die Anforderungen der beiden Größenkriterien in eine einzelne Kennzahl, die für gute Cluster möglichst dicht an 0 liegt und für ungeeignete Cluster auch größer als 1 sein kann.

Aus der Regularitätsfunktion $reg(k)$ und den einschränkenden Kriterien für die Zahl der Vertizes und geometrischen Abmessungen wird dann die Zielfunktion $Acc(k)$ kombiniert:

$$Acc(k) \;=\; reg(k) - regBound(k) \cdot sizeBound(k) \tag{9}$$

4.3 Erzeugung der Cluster

Das Erzeugen von Clustern kommt im Standardansatz nicht vor, sondern ist charakteristisch für die hier betrachtete Beschleunigungsstruktur. In anderen Gebieten der Computergrafik, etwa bei der Texturierung oder der Erzeugung mehrerer Detailgrade eines Meshes sind Verfahren, die Dreiecksnetze zu Clustern vereinfachen, aber bereits erprobt. Zwei im Ansatz unterschiedliche Algorithmen wurden für das hier benötigte Clustern erprobt: Das *iterative Wachstum*, und die *iterative Kontraktion*.

Iteratives Wachstum

Hier wird ein Dreieck aus dem Mesh ausgewählt, das dann einen $k = 1$-Cluster bildet. Für jedes an diesen Cluster angrenzende Dreieck d wird nun die Zielfunktion $Acc(k + d)$ berechnet. Das Dreieck mit dem höchsten positiven $Acc(k + d)$ wird dem Cluster hinzugefügt. Dieser Prozess wird iteriert. Ist $Acc(k + d) < 0$ für alle Nachbarn, ist der Cluster vollständig, und ein neues Startdreieck wird gesucht.

Iterative Kontraktion

Die iterative Kontraktion (vgl. [3]) nutzt den Dualgraphen des Meshes, also eine Repräsentation des Dreiecksnetzes, in der die Dreiecke als Vertizes abgelegt sind und Kanten aneinander grenzende Dreiecke verbinden. Benachbarte Dreiecke d_i und d_j bilden ein potentielles Cluster $c_{ij} = d_i + d_j$, sodass der Kante der Wert der Zielfunktion $Acc(c_{ij})$ zugewiesen wird. In einer globalen Liste wird daraufhin die Kante mit dem geringsten $Acc(c_{ij})$ kontrahiert, also das Cluster tatsächlich gebildet und als Vertex an Stelle der Einzelkomponenten eingefügt (vgl. Abb. 5).

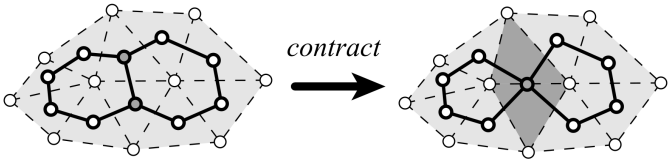

Abbildung 5: Kontraktion zweier Dreiecke zu einem Cluster. (Quelle: [3])

Iterativ werden benachbarte Cluster jeweils weiter zusammengefasst, bis im Graphen keine benachbarten Cluster mit positivem $Acc(c_{ij})$ mehr zu finden sind.

Vergleich der Methoden

Das iterative Wachstum baut einen Cluster nach dem anderen auf, sodass zu Beginn sehr reguläre Cluster entstehen, unter den verbleibenden Dreiecken dann aber zunehmend weniger geeignete Cluster geformt werden müssen. Die Laufzeit des Verfahrens ist linear in der Anzahl der Dreiecke und der Speicherbedarf begrenzt. Die iterative Kontraktion baut alle Cluster parallel auf. Damit sind die Unterschiede in der Regularität geringer, das Zusammenfassen von Clustern führt aber zu unregelmäßigeren Kantenverläufen und die globale Sortierung der $Acc(c_{ij})$ für die Kanten zu einer längeren Laufzeit. Somit eignet sich das iterative Wachstum eher für Szenen mit sehr vielen Dreiecken, während die iterative Kontraktion bei wenigen, möglicherweise in ihrer Größenverteilung unregelmäßigen Dreiecken ihre Stärken hat und deutlich bessere Ergebnisse liefert.

5 Ergebnisse

Um feststellen zu können, wie sich die Beschleunigungsstruktur auf den Renderaufwand auswirkt, wird sie gegen den Standardansatz getestet. Dafür müssen zunächst für die Parameter $MaxCount$ und $MaxSize$ des Clusteralgorithmus Werte gefunden werden, die die Laufzeit minimieren.

5.1 Bestimmung der idealen Clustergröße

Verglichen wird die Laufzeit für die Darstellung verschiedener Szenen mit den Parametern $MaxCount = MaxSize = 0$ (Standardverfahren; hier stellt jedes Dreieck einen Cluster dar), $MaxCount = MaxSize = 10$, $MaxCount = MaxSize = 50$ und $MaxCount = MaxSize = 100$.

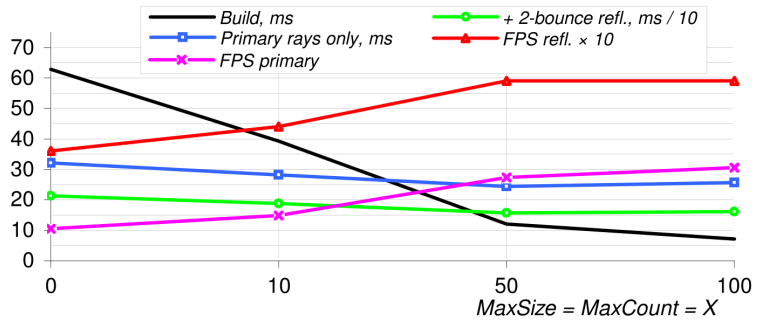

Abbildung 6: Laufzeit bei verschiedenen Parametern $MaxSize$, $MaxCount$ für Modell *Dragon* (252k Dreiecke). (Quelle: [1])

Abbildung 6 stellt für einen repräsentativen Fall die Laufzeiten der verschiedenen Renderschritte dar. Zu beobachten ist, dass der Aufwand zum Aufbau der BVH-Struktur mit steigender Clustergröße abnimmt, während der Aufwand für die Strahlverfolgung selbst bei $MaxSize = 50$ minimal ist und danach leicht wieder ansteigt. Dies ist das erwartete Verhalten für zu große Cluster, bei denen die BVH-Struktur zunehmend an Effizienz verliert. Basierend auf diesen Daten hält Garanzha den Wert 50 für die Parameter $MaxCount$ und $MaxSize$ fest und basiert darauf seine weiteren Tests.

5.2 Vergleich mit Referenzansätzen

Um die Vorteile aufzuzeigen, wird die Beschleunigungsstruktur nun mit anderen Verfahren verglichen. Folgende Verfahren kommen dabei zum Einsatz:

- *Standardansatz*: Kein Clustern. Jedes Dreieck wird einzeln mit seiner Bounding Box als Blatt des BVH-Baums abgespeichert. Dieses Verfahren entspricht den Parametern $MaxCount = MaxSize = 0$.
- *SAH-Cluster*: Clustern mit den beschriebenen Clusteralgorithmen (meist iterative Kontraktion) wird mit der Zielfunktion

$$Naive(k) = \sum_{i=1}^{k} \frac{SA(d_i)}{SA(BoundingBox(k))} \tag{10}$$

  durchgeführt. Diese Funktion maximiert die Clusteroberfläche pro Oberfläche der Bounding Box, achtet dabei aber nicht auf Regularität.
- *GWH01-Cluster*: Das von Garland[3] vorgestellte Clusterverfahren erzeugt in seiner Grundvariante Cluster, die sehr eben, aber nicht notwendigerweise kreisförmig sind. Der dortige Algorithmus wird mit den Parametern $a_1 = 0.2$, $a_2 = 2$ durchgeführt.

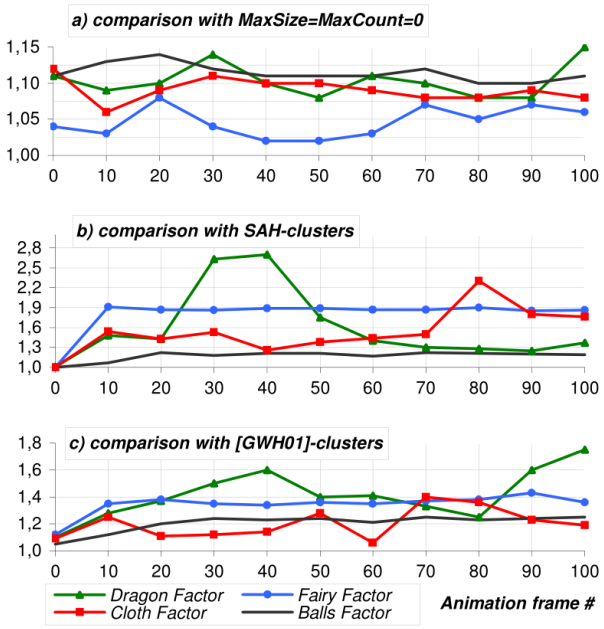

Abbildung 7: Leistungsfaktor (Laufzeit für das Referenzmodell, geteilt durch die Laufzeit für den neuen Ansatz) für die drei Referenzmodelle und vier verschiedene animierte Szenen im Verlauf der Animation. (Quelle: [1])

Abbildung 7 zeigt die Leistung des neuen Ansatzes im Vergleich mit den Referenzmodellen. Zu bemerken ist, dass die Renderzeit bei SAH- und GWH01-

Clustern im ersten Frame vergleichbar ist, aber für die späteren Frames deutlich anwächst. Das wird damit erklärt, dass diese Ansätze die Konnektivität eines Clusters nicht als Kriterium heranziehen und daher im Verlauf der Animation, wenn Cluster geringer Konnektivität auseinanderdriften, an Leistung einbüßen. Somit sind diese Clusterheuristiken für die weitere Animation sogar noch weniger performant als der Standardansatz, der gar kein Clustering durchführt. In jedem dargestellten Testfall führt die Nutzung des neuen Algorithmus zu einer geringeren Laufzeit. Auch im Vergleich mit grundlegend anderen Verfahren zur Beschleunigung animierter Szenen werden Ergebnisse gezeigt, nach denen der hier vorgestellte Ansatz eine signifikant bessere Performance bietet.

5.3 Einschränkungen der Anwendbarkeit

Die Anwendbarkeit des Algorithmus basiert auf Annahmen, deren Verletzung die Leistung stark negativ beeinflussen kann. So kann in Szenen, in denen sehr kleine Meshes mit wenigen Dreiecken vorkommen, keine effiziente Clusterbildung stattfinden. Zudem wird durch zu starke Verformung des Meshes die Regularität der Cluster eingeschränkt. Hier bieten sich Ansätze für eine weitere Optimierung des Verfahrens, indem etwa stark verformte Cluster bei Bedarf neu gruppiert werden. Die Umordnung von Clustern bei Bedarf ist auch geeignet, um hinzukommende oder verschwindende Dreiecke in der Szene zu behandeln. Da das Clustern nur die Blätter des BVH-Baums beeinflusst, sind Optimierungsansätze für Aufbau und Aktualisierung des Baumes selbst, oder gar die Wahl einer anderen Datenstruktur, unabhängig weiterhin möglich.

6 Fazit

Der von K. Garanzha vorgestellte Algorithmus zeigt einen Weg, das Rendern animierter Szenen durch Präprozessierung der dabei verwendeten Dreiecksnetze zu beschleunigen. Die vorgestellten Benchmarks zeigen, dass gegenüber anderen Ansätzen tatsächlich eine deutliche Beschleunigung festzustellen ist. Insbesondere können weitere Optimierungen bei Aufbau und Aktualisierung der räumlichen Datenstruktur in vielen Fällen übernommen werden, da der Algorithmus nicht auf eine spezifische Umsetzung dieser Struktur angewiesen ist. Allerdings basiert die hohe Leistung auf Annahmen über die Animation der Szene, die nicht immer vorausgesetzt werden können. Weiterer Aufwand ist hier nötig, um auch für allgemeine animierte Szenen eine entsprechende Beschleunigung zu erreichen.

7 Literatur

[1] Kirill Garanzha. The use of precomputed triangle clusters for accelerated ray tracing in dynamic scenes. *Computer Graphics Forum*, 28:1199–1206, 2009. doi: 10.1111/j.1467-8659.2009.01497.x.

[2] Kirill Garanzha. Vortrag: The use of precomputed triangle clusters for accelerated ray tracing in dynamic scenes. *20-th Eurographics Symposium on Rendering*, 2009. URL http://garanzha.com/Documents/UPTC-ART-DS.ppt. (abgerufen am 11.12.2011 14:56).

[3] Michael Garland, Andrew Wilmott, and Paul S. Heckbert. Hierachical face clustering on polygonal surfaces. *Proc. of the Symposium of Interactive 3D Graphics*, pages 49–58, 2001. URL http://mgarland.org/files/papers/cluster.pdf. (abgerufen am 11.12.2011 11:43).

[4] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray tracing animated scenes using motion decomposition. *Eurographics*, 25 Number 3:1–9, 2006.

[5] Houjun8022. Example of bounding volume hierarchy, December 2009. URL http://commons.wikimedia.org/wiki/File:Example_of_bounding_volume_hierarchy.JPG. (abgerufen am 17.12.2011 19:46).

[6] Ingo Wald. On fast construction of SAH-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE/EG Symposium on Interactive Ray Tracing. IEEE*, pages 33–40, 2007.

[7] Ingo Wald, William R. Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. State of the art in ray tracing animated scenes. In Dieter Schmalstieg and Jirí Bittner, editors, *STAR Proceedings of Eurographics 2007*, pages 89–116, Prague, Czech Republic, September 2007. Eurographics Association. ISBN 1017-4656.