

Kollisionserkennung mit Oriented Bounding Boxes und OBB-Trees

von

Sebastian Sippl

Bakkelaureatsarbeit

Salzburg, 28. Februar 2008

Überblick

Kollisionserkennung spielt eine große Rolle im Bereich der Computergrafik, unter anderem in Echtzeit-Anwendungen. Sogenannte Hüllkörper und Hüllkörperhierarchien finden heute eine breite Anwendung in verschiedenen Bereichen der Computergrafik. Dieses Dokument gibt einen kurzen Überblick der am häufigsten verwendeten Hüllkörper mit Fokus auf orientierte Hüllquader. Verschiedene Möglichkeiten zur Berechnung von orientierten Hüllquadern (Oriented Bounding Boxes) werden vorgestellt. Da dazu einige Fachbegriffe notwendig sind, werden diese in einem eigenen Kapitel behandelt. Die Standard-Methode, um zu überprüfen, ob sich orientierte Quader überlappen, wird besprochen - das "Separating Axis Theorem". Dabei wird auf einige Ideen zur Optimierung dieser Methode eingegangen. Danach wird die Anwendung dieses Theorems in einer hierarchischen Struktur präsentiert. Schließlich folgt ein kurzer Vergleich von orientierten Hüllquadern und deren Hierarchien mit achsenorientierten Hüllquadern, Kugeln und deren Hierarchien.

1

Inhaltsverzeichnis

1	Index	3
2	Einleitung	4
2.1	Hüllkörper/Hüllkörperhierarchien: Wozu ?	4
2.2	Kriterien bei der Hüllkörperwahl	4
3	Arten von Hüllkörpern	5
3.1	Kugeln, Kugel-Bäume	5
3.2	Achsenorientierte Quader, AABB-Bäume	6
3.3	K-Dops, K-Dop Bäume	7
4	Orientierte Hüllquader (Oriented Bounding Boxes)	8
4.1	Definition	8
4.2	Berechnung von OBBs	8
4.3	OBB-Schnitttests (Separating Axis Theorem)	12
5	Hierarchien aus OBBs (OBB-Trees)	15
5.1	Einleitung	15
5.2	Definition	15
5.3	SAT-Lite	15
5.4	Aufbauen der Hierarchie	15
5.5	Tests zwischen OBB-Trees	16
5.6	Konstruktion von OBB-Trees für Charaktermodelle	17
6	Vergleich von OBBs/OBB-Trees mit anderen Verfahren	18
6.1	Kollisionserkennungs-Kosten-Gleichung	18
6.2	“Separating Axis” vs. andere Methoden für OBBs	18
6.3	“Separating Axis” vs. AABB und Sphere Tests	18
6.4	Vergleich von OBB-Hierarchien mit AABB-Trees und Sphere-Trees	18
7	Mathematische Grundlagen für die Berechnung von Oriented Bounding Boxes	19
7.1	Hauptkomponentenanalyse (Principal Component Analysis)	19
7.2	Eigenvektoren	19
7.3	Kovarianzmatrix	20
7.4	Erwartungswerte	20
7.5	Berechnen der Kovarianzmatrix im Falle von Punkten	21
7.6	Linienintegrale von parametrisierten Linien	21
7.7	Berechnen der Kovarianzmatrix im Falle von Liniensegmenten	21
7.8	Vektorielle Flächenintegrale von parametrisierten Dreiecken	22
7.9	Berechnen der Kovarianzmatrix im Falle von Dreiecken	22

2 Einleitung

2.1 Hüllkörper/Hüllkörperhierarchien: Wozu ?

Das grundlegende Problem besteht darin, herauszufinden, ob sich zwei aus Dreiecken bestehende Objekte zu einer bestimmten Zeit überlappen. Ohne Einschränkung könnte das unter Umständen zu einer relativ großen Anzahl von Dreiecksschnitttests führen, da mit jedem Dreieck des einen Modells ein Schnitttest mit jedem Dreieck des anderen Modells durchgeführt werden müsste. Außerdem würde das zu großen Unterschieden für die benötigte Zeit der Schnitttests führen, wenn sich z.B. beim ersten Test sofort zwei Dreiecke schneiden würden und bei einem nachfolgenden Test erst die letzten. Solch ein Verhalten ist bei heutigen Szenen mit einer großen Anzahl von Polygonen/Dreiecken vor allem im Bereich von Echtzeit-Anwendungen undenkbar.

Die Lösung für dieses Problem besteht darin, diese Modelle in primitive geometrische Körper zu hüllen, um die Anzahl der Tests einzuschränken. Sollte man bei so einem Test eine Überlappung feststellen bedeutet das natürlich nicht, daß sich die Objekte wirklich überlappen. Das heißt wiederum, man müsste nun alle Dreiecke der beiden Objekte auf Überlappung testen. Aus diesem Grund untergliedert man die Objekte weiter in hierarchische Strukturen dieser Hüllkörper, um kostbare Rechenzeit zu sparen. Die Überlappungstests werden dann mit den Unterobjekten rekursiv durchgeführt. So können die benötigten Dreieckstests auf relativ wenige reduziert werden, sofern eine solche Genauigkeit überhaupt notwendig ist. Es ist natürlich auch möglich, so lange zu testen bis eine vorgegebene Rechenzeit überschritten ist.

2.2 Kriterien bei der Hüllkörperwahl

Die Haupt-Kriterien bei der Wahl von Hüllkörpern sind, wie genau ein Objekt von einem Hüllkörper repräsentiert werden kann und wieviel Zeit es kostet einen Überlappungstest für diese Art von Körper durchzuführen. Zum Beispiel kann ein langes, dünnes Objekt (ein Stab, Kugelschreiber) relativ schlecht von einer Kugel repräsentiert werden. Das Volumen der Kugel ist in jedem Fall ein Vielfaches größer als das Volumen dieses Objekts. Die hierarchische Unterteilung reduziert zwar das Volumen, aber je nach gewünschter Genauigkeit wird die hierarchische Struktur relativ tief (viele Knoten) werden. Es wäre also wesentlich sinnvoller, so ein Objekt mit einem orientierten Quader einzuhüllen. Dieser würde das Volumen des Stabes relativ genau treffen.

Ein weiteres wichtiges Kriterium ist die Komplexität der Tests. Also wie viele Rechenoperationen durchgeführt werden müssen, um auf Überlappung zu testen. Dies muss immer im Verhältnis zum Volumen des Hüllkörpers gesehen werden. Analog zum obigen Beispiel würden möglicherweise zwei orientierte Quader reichen, um zwei Stäbe einzuhüllen. Um diese Stäbe mit Kugeln einzuhüllen, müsste man mindestens eine Rekursionsstufe tiefer gehen, um eine ähnliche Genauigkeit wie mit den orientierten Quadern zu erreichen. Dies würde zu einer erhöhten Anzahl von Tests, im Vergleich zu nur einem Test im Falle von orientierten Quadern, führen.

Sollen die Hüllkörper in Echtzeit-Anwendungen Verwendung finden, ist es auch noch wichtig, ob die Körper bei Rotationen und Translationen neu berechnet werden müssen, oder ob es möglich ist, diese zusammen mit den Objekten zu transformieren.

3 Arten von Hüllkörpern

Dieser Abschnitt gibt einen kurzen Überblick über die zur Zeit am meist verbreiteten Hüllkörper, bzw. Hüllkörperhierarchien. Orientierte Quader werden hier bewusst weggelassen, da sie ohnehin später im Detail behandelt werden.

3.1 Kugeln, Kugel-Bäume

Kugeln (Spheres) werden oft als Hüllkörper benutzt, da Überlappungstests zwischen ihnen schnell zu berechnen sind. Eine Kugel besteht aus einem Radius r und einem Mittelpunkt m . Seien nun K_1 und K_2 zwei Kugeln, r_1, r_2 deren Radii und d der Abstand zwischen m_1 und m_2 , den Mittelpunkten der Kugeln. Die Kugeln überschneiden sich, wenn gilt:

$$d \leq r_1 + r_2$$

Für das Finden der Kugeln gibt es verschiedene Algorithmen. Je nach benötigter Genauigkeit muss man hier Kompromisse zwischen Geschwindigkeit des Algorithmus und Passgenauigkeit der Kugel eingehen. Ein einfacher, schneller Algorithmus funktioniert folgendermaßen: Man berechnet den achsenorientierten Quader der Polygonmenge, für die man eine Hüllkugel berechnen möchte. Von diesem Quader fungiert der Mittelpunkt als Mittelpunkt der Kugel, und die Diagonale des Quaders als Durchmesser. Ritter[10] und Welzl[12] präsentieren etwas komplexere Algorithmen um Hüllkugeln zu finden.

Kugeln werden auf Grund der Schnelligkeit des Tests oft auch zusätzlich zu anderen Tests implementiert. Ein weiterer Vorteil von Kugeln besteht darin, dass Kugeln im Bezug auf Rotationen und Translationen invariant sind, dh. sie verändern ihre Form und Größe nicht. Deshalb müssen aus Kugeln bestehende Bäume (Sphere Trees) nicht neu berechnet werden, wenn die darunterliegenden Objekte rotiert oder transliert werden. Die Transformation kann einfach auf die anfangs berechneten Mittelpunkte angewandt werden.

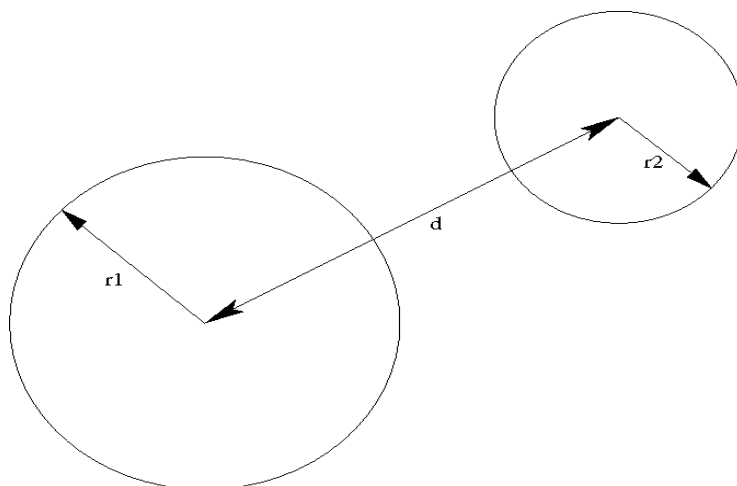


Abbildung 1: Überlappungstest von Kugeln, hier vereinfacht im 2D als Kreise dargestellt

3.2 Achsenorientierte Quader, AABB-Bäume

Achsenorientierte Quader, oft als AABB (Axis Aligned Bounding Box) abgekürzt, sind eine weitere, einfache Möglichkeit für Hüllkörper. Der Vorteil ist hier die einfache Berechnung und der relativ einfache Test.

Um den Achsenorientierten Quader einer Polygon-Menge zu finden, sucht man die minimalen und maximalen Werte der x,y und z Ausdehnung der Polygon-Menge. So findet man für einen Achsenorientierten Quader also 6 Werte:

$$x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$$

Möchte man nun für zwei AABBs S und P herausfinden, ob sie sich überlappen, muss man folgende Terme überprüfen:

$$(Sx_{min} > Px_{min}) \vee (Px_{min} > Sx_{max})$$

$$(Sy_{min} > Py_{min}) \vee (Py_{min} > Sy_{max})$$

$$(Sz_{min} > Pz_{min}) \vee (Pz_{min} > Sz_{max})$$

Ist einer dieser Terme wahr, überlappen sich die Achsenorientierten Quader nicht.

Dieser Test ist etwas langsamer als der Kugel-Schnitttest, aber immer noch relativ schnell zu berechnen. In manchen Szenarios wird durch AABBs eine bessere Ausnutzung des Raumes erreicht als mit Kugeln. Ein Nachteil ist, dass die Achsenorientierten Quader nach Rotation der Objekte neu berechnet werden müssen. Dies kann man vermeiden, indem man die AABB etwas größer wählt. Also so, dass das Objekt bei einer Rotation nicht über die Grenzen des Quaders hinausgeht. Ein weiterer Nachteil ist, wie auch bei Kugeln, daß die Quader für gewisse Objekte nicht besonders geeignet als Hüllkörper sind. Für einen senkrecht stehenden Stab würde eine AABB gut passen. Dreht man den Stab jedoch um 45° , wird die AABB äußerst ungeeignet.

Erwähnenswert ist noch, dass es sich bei AABBs um eine spezielle Form von K-Dops handelt, und zwar um 6-Dops.

3.3 K-Dops, K-Dop Bäume

K-Dop steht für K-Discrete Oriented Polytope. Ein K-Dop wird durch $\frac{k}{2}$ (wobei k gerade ist) normierte Normalvektoren, $n_i, 1 \leq i \leq \frac{k}{2}$, definiert.

Hierbei sind mit jedem n_i zwei skalare Werte d_i^{min} und d_i^{max} , mit $d_i^{min} < d_i^{max}$ verbunden. Jedes dieser Tripel $(n_i, d_i^{min}, d_i^{max})$ beschreibt ein ‘‘Slab’’ (Platte), S_i , das das Volumen zwischen den zwei Ebenen

$$\pi_i^{min} : n_i \cdot x + d_i^{min} = 0$$

und

$$\pi_i^{max} : n_i \cdot x + d_i^{max} = 0$$

repräsentiert.

Die Schnittmenge aller solcher ‘‘Slabs’’, $\bigcap_{1 \leq i \leq \frac{k}{2}} S_i$ ist das Volumen des eigentlichen K-Dops.

Die Berechnung von K-Dops funktioniert ähnlich wie bei AABBs. Die Punkte des Objekts werden auf jeden Normalvektor projiziert. Die Extremwerte dieser Projektionen werden in d_i^{min} und d_i^{max} gespeichert. Diese zwei Werte definieren dann die kompaktesten ‘‘Slabs’’ in dieser Richtung. Alle solche Werte zusammen bilden das minimale K-Dop.

Zwei **durch die selben Normalvektoren** definierte K-Dops S_i^A und S_i^B schneiden sich nicht, wenn

$$\forall i \in \{1, \dots, \frac{k}{2}\} : S_i^A \cap S_i^B = \{\}$$

gilt.

Wobei:

$$(S_i^A \cap S_i^B = \{\}) \iff ((d_i^{B,min} > d_i^{A,max}) \vee (d_i^{A,min} > d_i^{B,max}))$$

Die Konstruktion von K-Dop Bäumen wird ausführlich in der Doktorarbeit von Klo-sowski [8] behandelt. Die Berechnung von K-Dops ist sehr schnell und der Test ist mit dem der AABBs vergleichbar. Die Passgenauigkeit hängt von der Wahl und Anzahl der Normalen (k) ab. Der Nachteil besteht darin, dass K-Dops bei jeder Rotation neu berechnet werden müssen. Weiteres kann man in Fachliteratur oder vorher erwähnter Doktorarbeit nachlesen.

4 Orientierte Hüllquader (Oriented Bounding Boxes)

4.1 Definition

Eine Oriented Bounding Box, kurz OBB, ist ein Quader, dessen Seitennormalen paarweise orthogonal sind. Zum Beispiel also eine rotierte AABB. Eine OBB, hier B genannt, wird beschrieben durch einen Mittelpunkt b^c und drei normierte Vektoren b^u , b^v und b^w , die die Richtungen der Seiten der OBB beschreiben. Die OBB ist in dem Koordinatensystem der drei Vektoren also ein achsenorientierter Quader (AABB). Zusätzlich werden noch drei sogenannte Halbdistanzen (engl. Half-lengths) benötigt, die die (positiven) Abstände vom Mittelpunkt b_c zu den jeweiligen Seiten darstellen. Diese werden hier als h_u^B , h_v^B und h_w^B bezeichnet.[1]

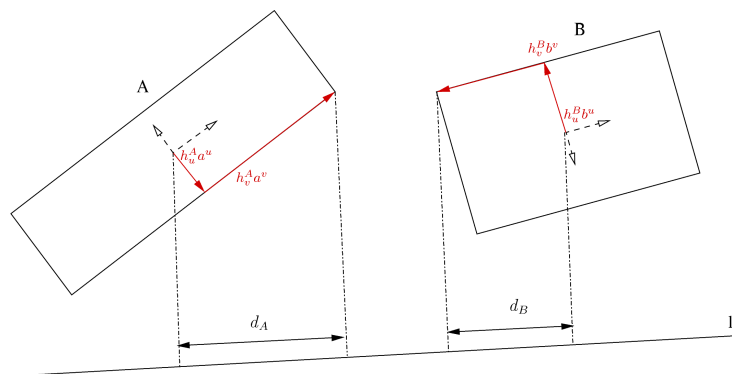


Abbildung 2: Darstellung der Halbdistanzen zweier OBBs und der Projektion der "Radii" auf eine Achse l .

4.2 Berechnung von OBBs

Die Berechnung von OBBs ist meist nicht ganz trivial. Es gibt dazu mehrere, verschiedene Methoden, bei denen es Unterschiede in der Genauigkeit und Geschwindigkeit der Berechnung gibt. Drei Methoden werden auf den nächsten Seiten präsentiert. Als erstes (und am genauesten) möchte ich auf die Methode von Gottschalk et al.[5] eingehen.

Berechnung von OBBs mittels PCA (Principal Component Analysis):

Die Hauptkomponentenanalyse (engl. Principal Component Analysis) extrahiert die wesentlichen Merkmale aus einer Wolke von Punkten.¹ Diese Methode eignet sich für die Berechnung der OBBs, da wir ja die Hauptachsen eines Objekts finden wollen. Ist das Objekt beispielsweise ein langes dünnes Objekt, wie ein Stift, dann möchten wir unsere OBB an der langen Achse des Stiftes ausrichten. An einem Blatt Papier z.B. würden wir eine Achse unserer OBB senkrecht zur Ebene des Papiers ausrichten wollen. Das kann erreicht werden, indem man die Hauptkomponenten der statistischen Verteilung der Geometrie untersucht. Ziel ist es, die Eigenvektoren einer Kovarianzmatrix zu finden. Diese entsprechen den Hauptkomponenten dieser Verteilung und bilden dann (normiert) das Koordinatensystem für die OBB. In diesem Koordinatensystem kann dann anhand der AABB-Methode die orientierte Box einfach berechnet werden, indem man die Maxima und Minima entlang den Hauptachsen sucht. Mit deren Hilfe kann dann auch der Mittelpunkt bestimmt werden.

Hierbei muss man, je nach Objekt, bestimmte Feinheiten beachten:

1. Die Konvexe Hülle der Punktwolke kann berechnet werden, da Punkte im Inneren des Objekts die Orientierung der Eigenvektoren unerwünscht beeinflussen. Das passiert, weil die Verteilung der Punkte verändert wird, und sich damit die Richtung der maximalen Ausdehnung der Punkte ändert.

Zwei Bilder dienen als Beispiel:

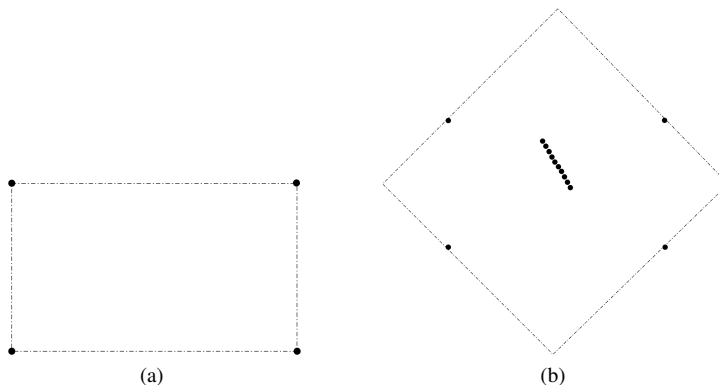


Abbildung 3: Interne Punkte verändern die Ausrichtung der OBB

Im ersten Bild (a) ist die Richtung der maximalen Ausdehnung mit dem Rechteck abgestimmt und die Passgenauigkeit der OBB ist perfekt. Im zweiten Bild (b) wird durch die zusätzlichen Punkte die Verteilung der Punkte und damit die Richtung der maximalen Ausdehnung so weit verändert, dass das Rechteck eine schlechte Passgenauigkeit erreicht.

Für die Berechnung der Konvexen Hülle (2D sowie 3D) kann man den QuickHull Algorithmus benutzen.

¹Die gesamten mathematischen Grundlagen dieser Methoden sind im letzten Kapitel zusammengefasst. Im letzten Kapitel befinden sich auch die verschiedenen Kovarianzformeln für die in diesem Abschnitt angeführten Methoden und die Grundlagen für deren Berechnung.

2. Die Verteilung der Extrempunkte kann die Richtung der Eigenvektoren beeinflussen. Wenn die Punkte der konvexen Hülle ausgeglichen verteilt sind, ist dies kein Problem. Sollten die Punkte jedoch nicht einheitlich verteilt sein, kann eine schlechte Ausrichtung der OBB stattfinden. Da jeder Punkt zur Richtung der maximalen Ausdehnung beiträgt, ist diese natürlich dort größer, wo mehr Punkte vorhanden sind. Zwei Bilder dienen hier wieder als Beispiel:

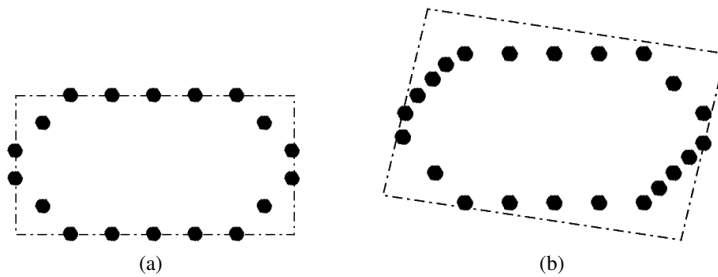


Abbildung 4: Ungleiche Verteilungen von Punkten beeinflussen die Orientierung der OBB

Dieses Problem kann nur dadurch gelöst werden, nicht die Verteilung von Punkten zu betrachten, sondern die Verteilung der unendlichen Ansammlung von Punkten, die die Primitiven (Linien, Dreiecke) selbst sind. Die Ecken links oben und rechts unten in den obigen Bildern besitzen zwar mehr Punkte, aber die Liniensegmente zwischen den Punkten sind in etwa gleich lang. Wenn man also eine Kovarianzmatrix mit Hilfe von Liniensegmenten berechnen kann, wird eine bessere Ausrichtung der OBB erreicht. Analog dazu kann man die Kovarianzmatrix auch mit Hilfe von Dreiecken berechnen. Allerdings erreicht auch mit dieser Methode die OBB nicht ihre optimale Passgenauigkeit.

Je nach Modell muss man also die passende Methode wählen.

Das Komplexeste hierbei ist die Berechnung der konvexen Hülle. Laut Gottschalk ist die Komplexität der Berechnung für die Kovarianzmatrix von n Punkten oder Dreiecken [5]:

Für Punkte und Dreiecksfacetten: $O(n)$

Für Facetten der Konvexen Hülle: $O(n \log n)$

Nach einer dieser Methoden berechnet man also die Kovarianzmatrix und deren Eigenvektoren. Diese werden dann normiert und bilden dann b^u , b^v , und b^w . Der Mittelpunkt b^c und die Halbdistanzen h_u^B , h_v^B und h_w^B können ebenfalls wie oben beschrieben berechnet werden und somit ist die OBB vollständig definiert.

Berechnung von fast orientierten Bounding Boxes(Almost Oriented Bounding Boxes):

Eine Methode, die ähnlich zu der vorhergehenden Methode ist, und Anwendung bei Computerspielen findet, wird von Ben St. John präsentiert[7]. Hierbei wird der Quader lediglich für die x und y Achse orientiert. Dadurch vereinfacht sich die Berechnung der Eigenvektoren und man spart Rechenzeit. Der Autor verwendet weiters eine vereinfachte konvexe Hülle aus den maximalen und minimalen x und y Werten (da die Box ja im 2D berechnet wird) und den Geraden $x + y = 0$ und $x - y = 0$. Hierbei kann, wie oben, ein Fehler durch die ungleiche Verteilung von Punkten entstehen. Laut dem Autor dieses Artikels ist der maximale dabei mögliche Fehler ca. 8% und wird durch eine Vergrößerung des Volumens um ca. 16% kompensiert. Zusätzlich wird eine Kugel als Hüllkörper benutzt und überprüft, ob diese vom Volumen her eine bessere Passgenauigkeit liefert.

Berechnung von OBBs nach Eberlys Methode:

Eberly [3] berechnet OBBs nach einer Minimierungsmethode. Mit seinem Algorithmus ist es nicht notwendig, die konvexe Hülle zu berechnen. Dieser Algorithmus ist iterativ. Aus diesem Grund hängt es von der ersten Schätzung ab, wie schnell sich die Lösung der minimalen OBB annähert. Eberly benutzt eine Menge aus den möglichen Richtungen der Achsen der OBB. Die Achsen der kleinsten OBB dienen als Startpunkt für die numerische Minimierung. Danach wird Powells [9] Methode benützt, um die minimale Box zu finden.

4.3 OBB-Schnitttests (Separating Axis Theorem)

Eine schnelle Methode zwei OBBs, hier A und B genannt, auf Überlappung zu testen, erfolgt durch das sogenannte Separating Axis Theorem[1][5]. Der Schnitttest wird im Koordinatensystem von A durchgeführt. Das bedeutet: Der Ursprung des Koordinatensystems befindet sich bei $a^c = (0,0,0)$ (a^c ist der Mittelpunkt von A). Die Hauptachsen des dreidimensionalen Koordinatensystems sind $a^u = (1,0,0)$, $a^v = (0,1,0)$ und $a^w = (0,0,1)$. Die OBB B befindet sich in einer zu A relativen Position, die durch eine Rotation R und eine Translation t erreichbar ist (R und t sind Matrizen).

Laut dem Separating Axis Theorem genügt es, eine Achse zu finden, die A und B räumlich voneinander trennt, um sicher zu sein, dass sie sich nicht überlappen. Dazu müssen insgesamt 15 Achsen getestet werden. Die drei Achsen, die orthogonal zu den Seiten von A sind, und die drei Achsen, die orthogonal zu den Seiten von B sind (6). Die restlichen 9 Achsen erhält man aus einer Kombination der Kanten von A und B durch Anwendung des Kreuzprodukts (Das Ergebnis des Kreuzprodukts von zwei Vektoren x und y liefert eine Achse, die orthogonal auf x und orthogonal auf y ist). Auf Grund der Orthonormalität der Matrix $A = (a^u, a^v, a^w)$ sind die Achsen, die orthogonal zu den Seiten von A sind, einfach die Achsen a^u, a^v und a^w . Das Gleiche gilt für die B.

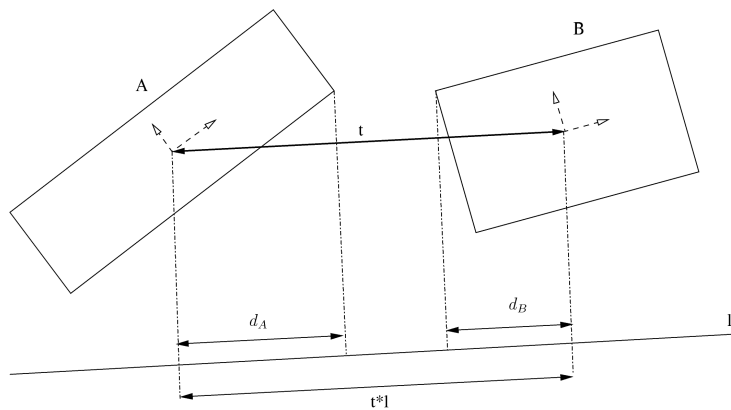


Abbildung 5: Darstellung des Separating Axis Theorems. Die OBBs überlappen sich nicht, wenn sich die Projektionen ihrer "Radii" auf eine der Achsen (hier l) nicht überschneiden.

Die Projektion der "radii" d_A und d_B erhält man durch folgende Formeln. Hierbei muss man den Betrag von h_i^A und h_i^B nicht berechnen, da bei den Halbdistanzen immer die positiven Werte gespeichert werden.

$$d_A = \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot l|$$

$$d_B = \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot l|$$

Wenn nun l eine trennende Achse ist, sind die Intervalle der Projektionen der "radii" auf diese Achse disjunkt.

Und es gilt:

$$|t \cdot l| > d_A + d_B$$

Es folgen Vereinfachungen dieser Gleichung für 3 Fälle. Die erste im Falle, dass l eine Kante von A ist. Die zweite Vereinfachung für den Fall, dass l eine Kante von B ist. Und die dritte Vereinfachung für den Fall einer Kombination von Kanten aus A und B.

Im Falle, dass l eine Achse von A ist, also z.B. für den Fall $l = a^u$ (Für die zwei anderen Fälle, also $l = a^v$ und $l = a^w$, funktioniert dies analog):

$$|t \cdot l| = |t \cdot a^u| = |t_x|$$

Wobei der letzte Schritt daher kommt, dass wir uns im Koordinatensystem von A befinden. Das bedeutet also $a^u = (1, 0, 0)$. In weiterer Folge werden für diesen Fall die Berechnungen von d_A und d_B vereinfacht:

$$\begin{aligned} d_A &= \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot l| = \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot a^u| = h_u^A \\ d_B &= \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot l| = \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot a^u| \\ &= h_u^B |b_x^u| + h_v^B |b_x^v| + h_w^B |b_x^w| = h_u^B |r_{00}| + h_v^B |r_{01}| + h_w^B |r_{02}| \end{aligned}$$

Der gesamte Test für $l = a^u$ wird somit zu:

$$|t_x| > h_u^A + h_u^B |r_{00}| + h_v^B |r_{01}| + h_w^B |r_{02}|$$

Das Ergebnis der Gleichung von d_A erhält man aus dem Grund der Orthonormalität der Matrix A. Wobei in den letzten Schritten gilt, dass

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix} = (b^u, b^v, b^w)$$

Da R die relative Rotationsmatrix ist.

Im Falle $l = b^u$ erhält man für:

$$\begin{aligned} |t \cdot l| &= |t \cdot b^u| = |t_x b_x^u + t_y b_y^u + t_z b_z^u| = |t_x r_{00} + t_y r_{10} + t_z r_{20}| \\ d_A &= \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot l| = \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot b^u| \\ &= h_u^A |b_x^u| + h_v^A |b_y^u| + h_w^A |b_z^u| = h_u^A |r_{00}| + h_v^A |r_{10}| + h_w^A |r_{20}| \\ d_B &= \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot l| = \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot b^u| = h_u^B \end{aligned}$$

Der gesamte Test für $l = b^u$ wird somit zu:

$$|t_x r_{00} + t_y r_{10} + t_z r_{20}| > h_u^A |r_{00}| + h_v^A |r_{10}| + h_w^A |r_{20}| + h_u^B$$

Den letzte Schritt bei der Berechnung von d_B erhält man wieder aus dem Grund der Orthonormalität von B und aus dem Grund, dass Einheitsvektoren vorliegen.

Die letzte Art von Test ist für den Fall, daß die Achse eine Kombination der Kanten von den beiden OBB ist. Er wird hier am Beispiel von $l = a^u \times b^v$ dargestellt und vereinfacht. Für die restlichen 8 Achsen kann man Tests analog berechnen. Der Test lautet also:

$$\begin{aligned}
|t \cdot l| &= |t \cdot (a^u \times b^v)| = |t \cdot (0, -b_z^v, b_y^v)| \\
&= |t_z b_y^v - t_y b_z^v| = |t_z r_{11} - t_y r_{21}| \\
d_A &= \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot l| = \sum_{i \in \{u,v,w\}} h_i^A |a^i \cdot (a^u \times b^v)| \\
&= \sum_{i \in \{u,v,w\}} h_i^A |b^v \cdot (a^u \times a^i)| = h_v^A |b^v \cdot a^w| + h_w^A |b^v \cdot a^u| \\
&= h_v^A |b_z^v| + h_w^A |b_y^v| = h_v^A |r_{21}| + h_w^A |r_{11}| \\
d_B &= \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot l| = \sum_{i \in \{u,v,w\}} h_i^B |b^i \cdot (a^u \times b^v)| \\
&= \sum_{i \in \{u,v,w\}} h_i^B |a^u \cdot (b^i \times b^v)| = h_u^B |a^u \cdot b^w| + h_w^B |a^u \cdot b^u| \\
&= h_u^B |b_x^w| + h_w^B |b_x^u| = h_u^B |r_{02}| + h_w^B |r_{00}|
\end{aligned}$$

Der gesamte Test für $l = a^u \times b^v$ wird somit zu:

$$|t_z r_{11} - t_y r_{21}| > h_v^A |r_{21}| + h_w^A |r_{11}| + h_u^B |r_{02}| + h_w^B |r_{00}|$$

Sobald also einer dieser Tests positiv ist, überlappen sich die OBBs nicht. Das bedeutet also, die restlichen Tests müssen nicht mehr durchgeführt werden, sobald die erste trennende Achse gefunden wurde. Gottschalk et al. weisen darauf hin, dass die Absolutbeträge von R vier mal benutzt werden und aus Effizienzgründen deswegen nur einmal berechnet werden müssen. Weiters ist es effizienter, die Tests in der hier beschriebenen Reihenfolge (Zuerst die Kanten von A, dann B, dann die Kombinationen) durchzuführen. Dies ist der Fall auf Grund der einfacher berechenbaren Tests der Achsen von A und deren Orthogonalität.[1]

Je nach Applikation kann es auch von Vorteil sein, zusätzlich einen Kugel-Schnitttest einzusetzen, da dieser kaum zusätzlichen Aufwand bedeutet und nicht überlappende Körper eventuell schnell verwirft.

5 Hierarchien aus OBBs (OBB-Trees)

5.1 Einleitung

Erstmals auf der SIGGRAPH 96 von Gottschalk et al.[4] als Paper mit dem Namen "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection " vorgestellt. Dies hat weitere Forschung im Bereich der Kollisionserkennungsalgorithmen stark beeinflusst. OBB-Tree wurde so entworfen, dass er speziell in Situationen in denen zwei Oberflächen sehr nahe beieinander und annähernd parallel sind gute Ergebnisse liefert.

5.2 Definition

Der Hüllkörper, der bei OBB-Trees Anwendung findet, ist, wie der Name schon sagt, ein orientierter Hüllquader. Ein Grund für die Entwicklung der OBB-Bäume war, daß durch OBBs grundsätzlich, wie schon erwähnt, eine wesentlich bessere Passgenauigkeit als bei AABBs oder Kugeln erreicht wird. Ein weiterer Grund war die Entwicklung des schon oben angeführten "Separating Axis Theorems", das wesentlich schneller war als die bisherigen Überlappungstest-Algorithmen.

5.3 SAT-Lite

Eine Verbesserung dieses Theorems speziell für OBB-Trees wurde von Van den Bergen vorgeschlagen [11]. Hierbei werden die letzten 9 Achsentests einfach übersprungen, da diese am meisten Zeit kosten und im Großteil der Fälle nur für wenige gefundene Überlappungen zuständig sind. Dadurch kann es natürlich manchmal passieren, daß zwei OBBs als überlappend erkannt werden, wenn sie sich nicht überlappen. Dies führt in wenigen Fällen dazu, dass weitere Rekursionsstufen nötig sind, um zu erkennen, ob wirklich eine Überlappung vorliegt. Im Großen und Ganzen wird jedoch die durchschnittliche Performanz verbessert. Hierzu gibt es ein Kollisionserkennungs-Package namens SOLID.

5.4 Aufbau der Hierarchie

Die Struktur des Baumes besteht aus einem Binärbaum, bei dem jeder Knoten eine OBB beinhaltet und die Blätter des Baumes aus einzelnen Dreiecken bestehen. Die Erfinder bauen diesen Baum nach einer "Top-Down"-Methode folgendermaßen auf: Als erstes wird eine OBB für die gesamte Menge von Dreiecken (des Objekts) berechnet, die die Wurzel des Baumes bildet. Danach wird die längste Achse dieser OBB anhand einer Ebene, die orthogonal auf dieser Achse steht, geteilt und zwar mit Hilfe der Koordinate des arithmetischen Mittels der Punkte in der OBB. So erhält man also eine Partition der Dreiecksmenge der vorhergehenden OBB. Die Dreiecke werden dann anhand ihrer Schwerpunkte den jeweiligen Partitionen zugewiesen. Sollte es aus dem Grunde, dass alle Dreiecksschwerpunkte auf der trennenden Ebene liegen, nicht möglich sein die Achse zu benutzen, werden die anderen Achsen nach abnehmender Länge probiert. Danach werden für die Dreiecke anhand ihrer Zuordnung zu den Partitionen neue OBBs berechnet. Dies kann dann weiter rekursiv durchgeführt werden. Gottschalk et al. weisen darauf hin, dass, wenn man statt des Arithmetischen Mittels der Punkte den Schwerpunkt der Dreiecke benützt, um die Achse zu teilen, ein balancierter Baum entsteht.[4]

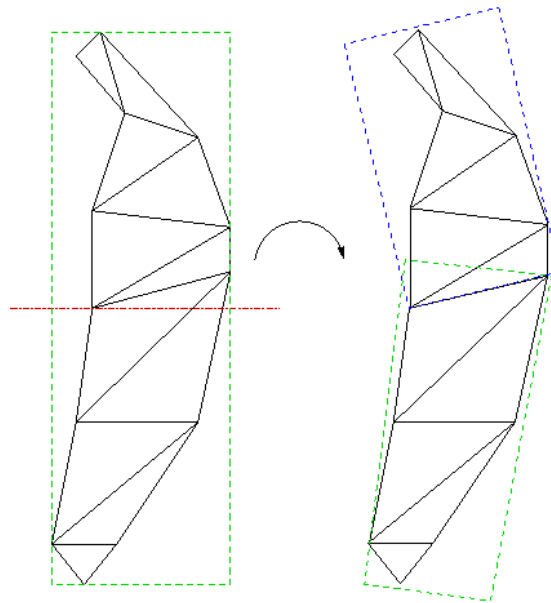


Abbildung 6: Darstellung des obigen Algorithmus. Die OBB wird anhand der längsten Achse geteilt, die Dreiecke den jeweiligen Partitionen zugeordnet und neue OBBs berechnet.

5.5 Tests zwischen OBB-Trees

In einem OBB-Baum wird jede OBB A zusammen mit einer Transformationsmatrix M_A (eine Rotationsmatrix R und einem Translationsvektor t) gespeichert. Diese Matrix beinhaltet die relative Orientierung und Position der OBB im Bezug auf ihren Vorgänger (oder die Transformation vom Ausgangskoordinatensystem bei der Wurzel).

Wenn man nun zwei OBB-Bäume auf Überlappung testen möchte, bei denen in der Wurzel die OBBs A und B stehen, geht man also wie folgt vor (wie im vorherigen Kapitel schon erwähnt): Angenommen der Überlappungstest wird im Koordinatensystem von A durchgeführt. Dazu muss man also B ins Koordinatensystem von A transformieren. Dies geschieht dadurch, zuerst B in die eigene Position (vom Ausgangskoordinatensystem aus) zu bringen, mit der Matrix M_B . Danach wird B ins Koordinatensystem von A transformiert, mit der Matrix M_A^{-1} . Zusammen also:

$$T_{AB} = M_A^{-1} M_B$$

Der OBB/OBB Überlappungstest (“Separating Axis Theorem”) benötigt als Parameter eine Rotationsmatrix R und einen Translationsvektor t, die die relative Orientierung und Position von B zu A beinhalten. Diese finden sich also in der Matrix:

$$T_{AB} = \begin{pmatrix} R & t \\ 0^T & 0 \end{pmatrix}$$

Angenommen, die beiden OBBs überlappen sich und man möchte mit einem der Knoten unter A, z.B. C, fortfahren. Wir entscheiden uns, den Test im Koordinatensystem von C durchzuführen. Das bedeutet also, B muss ins Koordinatensystem von C transformiert werden. Als Erstes wird B mit der Matrix T_{AB} ins Koordinatensystem von A

transformiert und danach weiter ins Koordinatensystem von C, mit M_C^{-1} . Diese Aufgabe erfüllt die folgende Matrix, die dann wie vorher für den Überlappungstest benutzt wird:

$$T_{CB} = M_C^{-1} T_{AB}$$

Diese Prozedur kann dann rekursiv für alle weiteren Tests genutzt werden. Der Vorteil besteht hier darin, dass die Matrix T_{AB} im Falle, dass eine Überlappung vorliegt, nicht neu berechnet werden muss.

5.6 Konstruktion von OBB-Trees für Charaktermodelle

Eine einfache Methode, um OBB Trees für Charaktermodelle zu erstellen, findet Anwendung bei Computerspielen. Die dabei entstehenden OBBs sind keineswegs minimal, erfüllen aber im Bezug auf Spiele ihren Zweck gut. Als Grundlage dient das Charakterskelett der Modelle, das normalerweise für animierte Charaktere ohnehin vorhanden ist.

Hierzu muss man verstehen, wie das sogenannte Vertex Blending funktioniert[1]:

Würde man ein Charaktermodell nur aus festen Objekten aufbauen, hätte man Probleme in den Bereichen der Gelenke. Diese würden unrealistisch aussehen, da sich die Objekte bei Bewegungen immer irgendwie überlappen würden. Es ist also besser, ein einziges, durchgängiges Objekt zu benutzen. Das Problem hierbei ist natürlich, dass der Körper dieses Objekts nicht flexibel ist. Die Lösung dafür liefert Vertex Blending (verschiedene Namen dafür sind existent ua. auch Skinning oä.). Nehmen wir als Beispiel einen Unter- und Oberarm. Wir definieren zusätzlich ein Charakterskelett. Die Knochen dieses Skeletts beeinflussen, mit Hilfe einer benutzerdefinierten Gewichtung, die Punkte des Körpers. Im Bereich des Unter- und Oberarms beeinflusst nur ein Knochen die Punkte des Körpers. Bei den Gelenken wird ein Teil vom Oberarm und ein anderer Teil vom Unterarm beeinflusst. Es ist auch möglich, Punkte im Bereich der Gelenke von beiden Knochen beeinflussen zu lassen. Diese Methode wird oft als Stitching (Vernähen) bezeichnet. Viele Kommerzielle Modelling Tools, wie z.B. 3DS-Max oder Maya, verfügen über solche Techniken.

Dieses Skelett ist hierarchisch aufgebaut, das bedeutet, jeder Knochen besitzt eine Matrix, M_i die seine Transformation im Bezug zum vorhergehenden Knochen widerspiegelt. Konkateniert man die vorhergehenden Matrizen, befindet man sich also im Koordinatensystem des jeweiligen Knochens.

Die Methode zum Berechnen von OBBs, anhand von Knochen, funktioniert nun wie folgt[6]:

Anfangs wird eingelesen, welche Knochen mit welchen Punkten assoziiert sind und auch deren Gewichtung. Diese Informationen werden dazu benutzt, im lokalen Koordinatensystem der Knochen, nach der AABB-Methode die minimalen und maximalen Punkte zu finden und somit eine OBB zu berechnen. Diese ist dann am jeweiligen Knochen orientiert und wird in der Datenstruktur des jeweiligen Knochens gespeichert. Findet eine Transformation des Knochens statt, wird die OBB einfach mit dem Knochen zusammen transformiert und braucht nicht neu berechnet zu werden. Diese Charakter-OBB-Bäume können natürlich auch mit nicht Charakter-OBB-Bäumen auf Überlappung getestet werden.

6 Vergleich von OBBs/OBB-Trees mit anderen Verfahren

In diesem Abschnitt wird ein Vergleich von OBBs und OBB-Trees, mit AABBs und AABB-Trees sowie mit Kugeln und Kugel-Bäumen präsentiert.

6.1 Kollisionserkennungs-Kosten-Gleichung

Eine Kollisionsabfrage besteht aus einer Anzahl von Hüllkörper-Überlappungstests, sowie möglichen Polygon Schnitttests. Es ist hilfreich, die benötigte Zeit für so eine Abfrage als Gleichung zu modellieren:

$$T = T_v N_v + T_p N_p$$

Wobei T_v die benötigte Zeit für einen Überlappungstest ist, N_v die Anzahl solcher Tests, T_p die Zeit für einen Polygon-Schnitttest und N_p die Anzahl solcher Tests. Weiters sind die Kosten für das Aktualisieren der Hüllkörper hier in T_v enthalten.

6.2 “Separating Axis” vs. andere Methoden für OBBs

Die Methoden, die für OBB Überlappungstests, neben dem “Separating Axis Theorem”, noch bekannt sind, sind der GJK Algorithmus und Seidels Lineare Programmierung (randomized linear programming algorithm). Laut den durchgeführten Benchmarks von Gottschalk et. al[5] ist das “Separating Axis Theorem” ca. 10 mal so schnell wie die GJK Methode und ca. 35 mal so schnell wie die Lineare Programmierung. Wobei Gottschalk et al. anmerken, dass diese Zahlen auf Grund der Spezialisierung des Theorems so hoch sind.

6.3 “Separating Axis” vs. AABB und Sphere Tests

In von den selben Personen durchgeführten Vergleichen für die Schnitttests von OBBs mit denen von AABBs und Kugeln sind folgende Ergebnisse angeführt: Im Falle, dass keine Überschneidung gefunden wurde, ist das “Separating Axis Theorem” etwas mehr als 5 mal so langsam wie der Kugel-Test und etwas weniger als 5 mal so langsam wie der AABB-Test. Im Falle, dass sich die Objekte schneiden, ist der Kugel-Test ca. 8 mal so schnell wie der OBB Test und der AABB Test etwas mehr als 5 mal so schnell.

6.4 Vergleich von OBB-Hierarchien mit AABB-Trees und Sphere-Trees

Laut Gottschalk et al. liegt der größte Vorteil von OBB-Trees, gegenüber AABBs und Kugeln in Situationen, in denen sich zwei Objekte streifen oder fast streifen.

- Für zwei große Modelle ist bei OBB-Trees in der obigen Gleichung T_v um eine Größenordnung langsamer als bei AABB-Trees und Sphere-Trees.
- N_v und N_p sind bei OBB-Trees asymptotisch niedriger als bei AABB oder Sphere-Trees.

Deswegen brauchen, sofern die Modelle groß genug sind und sie sich sehr nahe beieinander befinden, OBB-Trees weniger Abarbeitungszeit für die Kollisionserkennung als die anderen beiden Methoden.

7 Mathematische Grundlagen für die Berechnung von Oriented Bounding Boxes

Dieser Abschnitt gibt eine kurze Einführung der verschiedenen Begriffe, die für die Berechnung der orientierten Hüllquader benötigt werden. Ich halte mich dabei größtenteils an die Definitionen aus der Dissertation von S. Gottschalk [5] bzw. an die in mathematischen Fachbüchern enthaltenen Definitionen von Erwartungswerten etc.[2]. Ich verzichte bei verschiedenen Berechnungen auf einige Zwischenschritte. Diese kann man jedoch genauer in obiger Dissertation nachlesen und/oder mit Hilfe von Fachbüchern[2] selbst nachrechnen.

7.1 Hauptkomponentenanalyse (Principal Component Analysis)

Ziel der Hauptkomponentenanalyse ist es, die wesentlichen Merkmale einer Punktwolke zu extrahieren. Genauer gesagt, es soll ein neues Koordinatensystem so in eine Punktwolke hineingelegt werden, dass die Varianz in Richtung der Achsen jeweils maximiert wird. Dazu legt man zuerst eine Achse in der Richtung mit der maximalen Varianz in die Punktwolke. Die zweite Achse steht senkrecht auf die erste Achse, wieder so, daß die Varianz maximal ist usw. . Das Hineinlegen der Achsen entspricht der Berechnung der Eigenvektoren der Kovarianzmatrix dieser Punktwolke.

7.2 Eigenvektoren

Ein Eigenvektor ist ein vom Nullvektor verschiedener Vektor, der durch seine Abbildung die Richtung nicht verändert. Den Streckungsfaktor bezeichnet man als Eigenwert. Also:

$$Ax = \lambda x$$

A ist eine quadratische Matrix und $x \neq 0$. Wenn x die Gleichung erfüllt, ist x ein Eigenvektor. λ ist der zugehörige Eigenwert. Für die Berechnung von Eigenvektoren gibt es verschiedene Möglichkeiten. Eine Methode, um die Eigenvektoren einer Matrix zu berechnen, funktioniert wie folgt:

Als Erstes müssen die Eigenwerte berechnet werden. Dazu bildet man das charakteristische Polynom von A und berechnet dessen Nullstellen. Das Charakteristische Polynom von A ist wie folgt definiert:

$$p_A(\lambda) = \det(A - \lambda I) = 0$$

Nachdem man die Eigenwerte berechnet wurden, benutzt man diese im Gausschen Eliminationsverfahren, um die Eigenvektoren zu berechnen.

7.3 Kovarianzmatrix

Eine Ansammlung von Punkten im dreidimensionalen Raum bildet eine Punktwolke. Diese Punktwolke besitzt eine bestimmte statistische Verteilung und einen Schwerpunkt m . Die Kovarianzmatrix dieser Punkte liefert Information darüber, wie die Punkte im Raum verteilt sind. Z.B. kugelförmig oder auf einer Ebene. Die Eigenvektoren dieser Matrix liefern die Richtungen, in denen die Punktwolke die größte und kleinste statistische Ausdehnung besitzt. Die Kovarianzmatrix für einen n -dimensionalen Raum ist eine $n \times n$ Matrix.

Die Stelle i, j der Kovarianzmatrix ist definiert als:

$$C_{ij} = E(x_i x_j) - E(x_i)E(x_j)$$

$E()$ ist dabei der jeweilige Erwartungswert des Punktes x mit der Koordinate i oder j .

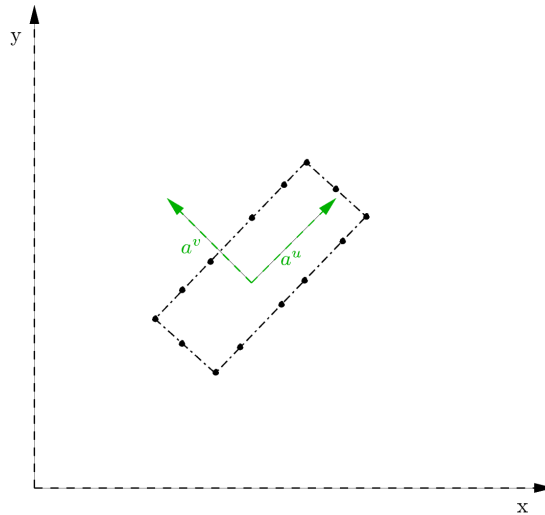


Abbildung 7: Eigenvektoren der Kovarianzmatrix, der dargestellten Punkte

7.4 Erwartungswerte

Um später die Kovarianzmatrix berechnen zu können, benötigt man also die Definition des Erwartungswerts. Die Erwartungswerte für einen Zufallsvektor, mit gleicher Verteilung, aus einer Menge von Punkten $p^1 \dots p^n$, in einer Dimension, (x,y oder z) sind definiert als:

$$E(x_i) = \frac{\sum_{k=1}^n p_i^k}{n}$$

$$E(x_i x_j) = \frac{\sum_{k=1}^n p_i^k p_j^k}{n}$$

Wobei x_i die i -te Koordinate des jeweiligen Punktes ist. $x = (x_x, x_y, x_z)$.

Im weiteren Verlauf muss der Erwartungswert für einen Zufallsvektor, der einen beliebigen Punkt aus einer Menge von Liniensegmenten oder Dreiecken, mit gleicher Wahrscheinlichkeit, annehmen kann, berechnet werden. Gleiche Wahrscheinlichkeit

bedeutet hier, dass die Wahrscheinlichkeit einen Punkt eines bestimmten Liniensegments zu wählen, proportional zur Länge des Segments ist. Für Dreiecke ist diese Wahrscheinlichkeit proportional zur Fläche des Dreiecks. Für solche kontinuierliche Zufallsvektoren sind die Erwartungswerte wie folgt definiert[5][2]:

$$E(x_i) = \frac{\int_M x_i dS}{\int_M dS}$$

$$E(x_i x_j) = \frac{\int_M x_i x_j dS}{\int_M dS}$$

Wobei M die Integrationsdomäne, die Ansammlung von Liniensegmenten oder Dreiecken ist.

7.5 Berechnen der Kovarianzmatrix im Falle von Punkten

Die Berechnung der Kovarianzmatrix für eine Menge von n Punkten lautet (aus der Definition des Erwartungswerts und der Kovarianzmatrix):

$$C_{ij} = \frac{1}{n} \sum_{k=1}^n [p_i^k p_j^k] - \frac{1}{n} \sum_{k=1}^n [p_i^k] \frac{1}{n} \sum_{k=1}^n [p_j^k]$$

7.6 Linienintegrale von parametrisierten Linien

Um in weiterer Folge die Erwartungswerte berechnen zu können, benötigt man noch die Definition von Linienintegralen parametrisierter Linien (Kurvenintegrale 1. Art). Seien p und q die Endpunkte einer Linie. Die Parameterdarstellung x(s) dieser Linie P lautet:

$$x(s) = p + s(q - p), s \in [0, 1]$$

Mit dieser Parametrisierung der Linie lautet das Linienintegral einer beliebigen Funktion über der Linie P, wie folgt:

$$\int_P f dS = L \int_0^1 f ds$$

Wobei L die Länge der Linie ist.

7.7 Berechnen der Kovarianzmatrix im Falle von Liniensegmenten

Eine Integration über M (die Ansammlung von den n Liniensegmenten des Modells) kann als Summe der Integrale über die einzelnen Liniensegmente , die im weiteren Verlauf mit P^k bezeichnet sind, dargestellt werden. Das bedeutet also:

$$\int_M f dS = \sum_k \int_{P^k} f dS$$

Weiters ist das Integral der Konstanten Funktion 1 eines Liniensegments die Länge dieses Segments P^k , die mit L^k bezeichnet wird.

$$\int_{P^k} dS = L^k$$

Die Summe der einzelnen Segmente, L^k , wird als L^M bezeichnet.

Durch Substitution und Umformungen entsteht aus den oben definierten Erwartungswerten:

$$E(x_i) = \frac{1}{L^M} \sum_k \int_{P^k} x_i^k dS$$

$$E(x_i x_j) = \frac{1}{L^M} \sum_k \int_{P^k} x_i^k x_j^k dS$$

Die restlichen Integrale werden mit Hilfe der parametrisierten Liniensegmente berechnet:

$$\int_{P^k} x_i^k dS = L^k m_i^k$$

$$\int_{P^k} x_i^k x_j^k dS = \frac{L^k}{6} (4m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k)$$

Wobei m^k der Mittelpunkt der k-ten Linie ist.

Aus diesen Berechnungen folgen die Erwartungswerte:

$$E(x_i) = m_i^M = \frac{1}{L^M} \sum_k L^k m_i^k$$

$$E(x_i x_j) = \frac{1}{6L^M} \left(\sum_k L^k (4m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k) \right)$$

m^M ist hierbei der Schwerpunkt der Gesamtheit der Linien.

Damit lässt sich nun die Kovarianzmatrix berechnen:

$$C_{ij} = \frac{1}{6L^M} \left(\sum_k L^k (4m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k) \right) - m_i^M m_j^M$$

7.8 Vektorielle Flächenintegrale von parametrisierten Dreiecken

Die alternative Methode, für Dreiecke, involviert die Berechnung von vektoriellen Oberflächenintegralen parametrisierter Dreiecke. Seien die Eckpunkte eines Dreiecks p , q und r . Die Parameterdarstellung $x(s, t)$ des Dreiecks P lautet:

$$x(s, t) = p + s(q - p) + t(r - p), s \in [0, 1], t \in [0, 1 - s]$$

Das vektorielle Flächenintegral einer Funktion $f(s, t)$ über der Fläche dieses Dreiecks P lautet:

$$\int_P f dA = |(q - p) \times (r - p)| \int_0^1 \int_0^{1-s} f(s, t) dt ds$$

7.9 Berechnen der Kovarianzmatrix im Falle von Dreiecken

M ist hier die Ansammlung von den n Dreiecken des Modells. Wie bei den Liniensegmenten kann man hier die Integrale in die Summen der Integrale der einzelnen Dreiecke, hier wieder als P^k bezeichnet, aufteilen. Zusätzlich gilt, ähnlich wie bei den Liniensegmenten: Das Integral einer konstanten Funktion 1 eines Dreiecks P^k ist die Fläche A^k dieses Dreiecks k. Also:

$$\int_{P^k} dA = A^k$$

Somit lassen sich die Erwartungswerte umformen auf:

$$E(x_i) = \frac{1}{A^M} \sum_k \int_{P^k} x_i^k dA$$

$$E(x_i x_j) = \frac{1}{A^M} \sum_k \int_{P^k} x_i^k x_j^k dA$$

Wobei A^M die Fläche der Gesamtheit von Dreiecken bildet. Mit Hilfe der parametrisierten Dreiecke kann man wieder die restlichen Integrale berechnen:

$$\int_{P^k} x_i^k dA = A^k m_i^k$$

$$\int_{P^k} x_i^k x_j^k dA = \frac{2A^k}{24} [9m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k + r_i^k r_j^k]$$

m^k bezeichnet hier den Schwerpunkt des k-ten Dreiecks.

Somit erhalten wir für die Erwartungswerte:

$$E(x_i) = m_i^M = \frac{1}{A^M} \sum_k A^k m_i^k$$

$$E(x_i x_j) = \frac{1}{A^M} \sum_k \frac{A^k}{12} [9m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k + r_i^k r_j^k]$$

m^M ist hier der Schwerpunkt der Gesamten Dreiecke.

Damit lässt sich nun wieder die Kovarianzmatrix formulieren:

$$C_{ij} = \frac{1}{A^M} \sum_k \frac{A^k}{12} [9m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k + r_i^k r_j^k] - m_i^M m_j^M$$

Literatur

- [1] Tomas Akenine-Möller and Eric Haines. *Real Time Rendering*. A K Peters, 2002.
- [2] I.N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 2005.
- [3] John Eberly. *3D Game Engine Design: A Practical Approach to Real Time Computer Graphics*. Morgan Kaufmann Publishers Inc., 2000.
- [4] S. Gottschalk, M. C. Lin, and D. Manocha. „*OBTree: A Hierarchical Structure for Rapid Interference Detection*“. Computer Graphics (SIGGRAPH 96 Proceedings), 1996.
- [5] Stefan Gottschalk. *Collision Queries using Oriented Bounding Boxes*. Ph.D. Thesis, Chapel Hill, 2000.
- [6] Oliver Heim, Carl S. Marshall, and Adam Lake. „Fast Collision Detection for 3D Bones-Based Articulated Characters“. In Andrew Kirmse, editor, *Game Programming Gems 4*. Charles River Media, 2004.
- [7] Ben St. John. „Enhanced Object Culling with (Almost) Oriented Bounding Boxes“. In Michael Dickheiser, editor, *Game Programming Gems 6*. Charles River Media, 2006.
- [8] James T. Klosowski. *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments*. Ph.D. Thesis, State University of New York at Stony Brook, May 1998.
- [9] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. „*Numerical Recipes in C*“. Cambridge University Press, 1992.
- [10] Jack Ritter. „An Efficient Bounding Sphere“. In *Graphics Gems*, pages 301–303. Academic Press, 1990.
- [11] Gino van den Bergen. *Collision Detection in Interactive 3D Computer Animation*. Ph.D. Thesis, Eindhoven University of Technology, 1999.
- [12] Emo Welzl. „Smallest Enclosing Disks (Balls and Ellipsoids)“. In H. Maurer, editor, *New Results and New Trends in Computer Science*. LNCS 555, 1991.