Kollisionserkennung - Broad Phase

Alexander van Renen

Fakultät für Informatik
Technische Universität München
Boltzmannstr. 3
85748 Garching, Deutschland
renen@in.tum.de

21. Juli 2012

Zusammenfassung

Kollisionserkennung wird benötigt um in Spielen oder anderen Anwendungen, wie physikalischen Simulationen, zu erkennen ob sich zwei Körper überlappen. In der "Broad Phase" wird versucht möglichst viele potentielle Kollisionen auszuschließen, um die Anzahl an durchzuführenden Tests zu verringern. Dieses Ziel wird erreicht durch das Verwenden von "Bounding Boxen" und Space "Partitioning Schemen", wie reguläre Gitter oder Bäume.

Inhaltsverzeichnis

1 Einführung

In einer Zeit mit Prozessoren, die meist mehrere Kerne im Giga-Herz Bereich besitzen, wird es für viele Anwendungen stets unwichtiger an jeder Stelle die effizientesten und trickreichesten Algorithmen und Datenstrukturen zu verweden. Ausnahmen dazu sind vorallem Berechnungen, Server- oder Echtzeitanwendungen.

Auf dem hart umkämpften Mark der Computerspiele ist dies nicht anders, jedes neue Spiel muss sich auf irgendeine Weise von den anderen abheben um Erfolg zu haben. Dies wird oft durch immer realistischere Graphik, Physik oder durch die schiere Größe der Spielwelt erreicht. Um diese zu ermöglichen müssen die zur Verfügung stehenden Ressourcen – wie Rechenleistung und Speicher – möglichst effizient genutzt werden.

Ein kritisches System ist hier die Kollisionserkennung, welche im wesentlich die Frage, ob sich zwei Objekte in der Spielwelt zu einem gegebenen Zeitpunkt berühren, beantwortet. Ein naiver Ansatz wäre eine Implementierung die jedes Objekt gegen jedes andere Objekt testet, was offensichtlich mit $O(n^2)$ skaliert und damit für große Datenmengen undenkbar ist.

Stattdessen versucht man die Objekte in der Spielwelt in disjunkte räumliche Gebiete zu partitionieren. Das hat zur Folge, dass man jeweils nur die Objektgruppen in einer Partition auf Kollision testen muss. Dieses Verfahren nennt sich Space Partitioning und wird in 3 ausführlich behandelt.

Der Inhalt von Kapitel 1 beschäftigt sich mit Bounding Volumes. Hier umhüllt man jedes Objekt der Spielwelt mit einem einfachem geometrischem Körper – einem Bounding Volume. Wenn sich die Bounding Volumes zweier Objekte nicht überschneiden so können sich auch die zugrundeliegenden Objekte nicht überschneiden. Auf diese Weise kann man den schwierigen – und damit langsamen – Kollisionstest zwischen zwei Objekten auf einen simpleren – und damit schnelleren – reduzieren.

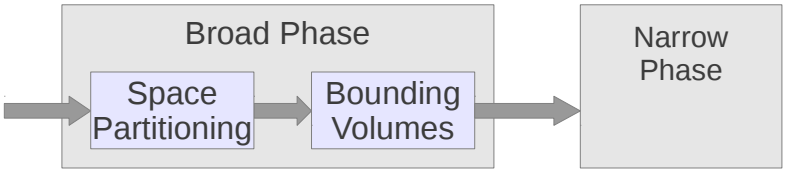

Abbildung 1: Kollisionserkennung teilt sich in die Broad und Narrow Phase auf.

Das Space Partitioning und die Verwendung von Bounding Volumes bilden zusammen die Grundlage der so genannten Broad Phase. Diese findet in der Spielwelt eine möglichst kleine Anzahl von Objektgruppen die sich möglichst wahrscheinlich überschneiden. Die endgültige Auswertung ob die Objekte innerhalb der Gruppen sich wirklich überscheiden wird auf Basis der Geometrie der Objekte gemacht und ist Inhalt der Narrow Phase. Eine Übersicht ist in Abb. 1 zu finden. Die Narrow Phase wird hier nicht behandelt, eine Einführung kann in [3] gefunden werden.

Es sei vermerkt, dass die hier eingeführten Verfahren alle an zwei dimensionalen Beispielen erklärt werden. Diese Wahl wurde getroffen, da zwei dimensionale Beispiele sich auf einem zwei dimensionalem Medium besser darzustellen lassen und da in allen Fällen die Erweiterung um eine Dimension – oder sogar mehrere Dimensionen – keine Probleme macht.

2 Bounding Volumes

Zu den typische Objekte in Spielen gehören Menschen, Bäume, Häuser und ähnliches. All diese Objekte haben komplizierte geometrische Formen, was einen Kollisionstest teuer, im Bezug auf die Laufzeit, macht [1]. Um diesen teuren Test so selten wie möglich ausführen zu müssen führt man Bounding Volumes ein. Ein Bounding Volume ist ein simpler geometrischer Körper der das komplizierte Objekt umschließt.

Um zu testen ob sich Objekt A und B überschneiden wird zuerst getestet ob sich ihre Bounding Volumes überscheiden. Ist das nicht der Fall dann ist es für die Objekte A und B unmöglich sich zu überlappen und man kann auf den teuren Kollisionstest verzichten. Nur wenn sich die Bounding Volumes überlappen ist es notwendig die eigentliche Geometrie der Objekte zu testen.

Aus diesen Beobachtungen folgen die beiden wesentlich Eigenschaften die ein gutes Bounding Volume ausmachen:

- Zum einen ist wichtig, dass sich das Bounding Volume eng um den eigentlichen Körper schmiegt. Damit wird die Wahrscheinlichkeit, dass sich die Bounding Volumes zweier Körper überlappen ohne das sich die eigentlichen Körper überlappen reduziert. Womit mehr unnötige Kollisionstests vermieden werden können.

[1] Wenn im Folgendem über teure und billitge Algorithmen geredet wird ist dies – wie auch hier – stets auf die Laufzeit und nicht auf deren Preis bezogen.

- Zum anderen ist ein weiteres, mit der ersten Eigenschaft konkurrierendes, Optimierungsziel das Erstellen eines möglichst einfachen Bounding Volumes. Je einfacher das Bounding Volume, desto schneller ist der Kollisionstest zwischen zwei Bounding Volumes. Allerdings umschließen einfache Bounding Volumes ihre Körper in der Regel schlechter als kompliziertere. Hier ist also abzuwägen.

2.1 Arten von Bounding Volumes

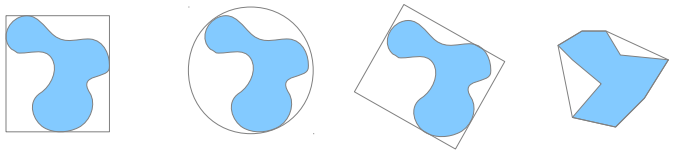

Abbildung 2: Verschiedene Arten von Bounding Volumes.

In Abbildung 2 sind einige klassische Vertreter von Bounding Volumes gezeigt. Diese sollen im Folgenden kurz beschrieben werden (von links nach rechts).

- Eine einfache Art eines Bounding Volumes stellt die Axis Aligned Bounding Box (AABB) dar. Hierbei handelt es sich um ein an den Koordinaten Achsen ausgerichtetes Rechteck. Seine Form kann durch das Speichern von zwei gegenüberliegenden Eckpunkten beschrieben werden. Der Kollisionstest kann mit Hilfe des Separating Axis Theorem (in 2.2 beschrieben) sehr effizient gelöst werden. Wird der eigentliche Körper gedreht, so muss die Axis Aligned Bounding Box mitgedreht werden.

- Das zweite Bild in Abbildung 2 zeigt einen Bounding Circle. Ein Kreis kann durch Angabe des Mittelpunktes und dem Radius gespeichert werden, was ihn sehr Speichereffizient macht. Für den Kollisionstest muss lediglich überprüft werden ob die Distanz der Mittelpunkte kleiner ist als die Summe der Radien. Bei der Rotation des Körpers muss der Bounding Circle nicht verändert werden sofern der Rotationsmittelpunkt des Körpers mit dem Mittelpunkt des Bounding Circles übereinstimmt.

- Auf den letzten beiden Bildern ist eine Oriented Bounding Box (OBB) und eine Convex Hull, welche sich beide ähnlich zu einer normalen, Axis Aligned Bounding Box, verhalten, gezeigt. Bei beiden müssen zusätzliche Parameter gespeichert werden, wie die Drehung und die zusätzlichen Ecken. Für den Kollisionstest kann, analog zur Axis Aligned Bounding Box, das Separating Axis Theorem verwendet werden. Die Rotation des eigentlichen Körpers kann analog zum Bounding Circle behandelt werden.

2.2 Separating Axis Theorem

Das Separating Axis Theorem dient als grundlegende Technik um die Kollision von Konvexen n-Ecken zu behandeln. Es kann für Axis Aligned Bounding Boxen, Oriented Bounding Boxen und Convex Hulls verwendet werden.

Theorem 1 (Separating Axis) *Zwei konvexe Körper schneiden sich genau dann nicht, wenn es eine Achse gibt, auf der sich ihre Projektionen nicht nicht schneiden. Es ist ausreichend alle Achsen, die durch die Normalen der Seiten der Körper besimmt sind, zu testen.*

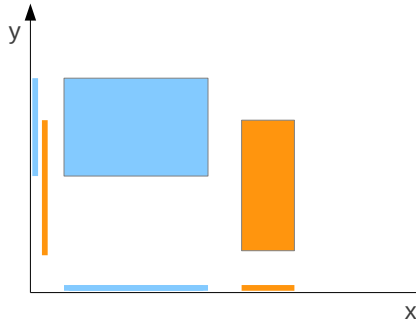

Abbildung 3: Keine Kollision

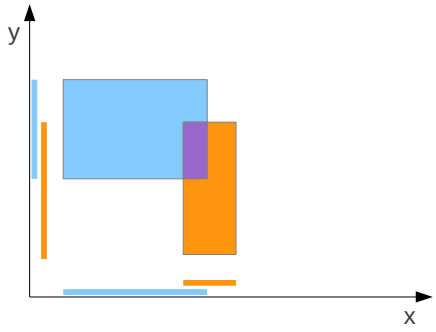

Abbildung 4: Kollision

Das vorgehen ist wie in dem Theorem beschrieben und soll im Folgenden exemplarisch an den Axis Aligned Bouning Boxen gezeigt werden. Für weitere Details, die Anwendung auf andere Bounding Volumes und Hinweise für die Implementierungs sei auf [2] verwiesen. Eine Visualisierung dieses Problems ist in in Abbildung 3 und 4 zu finden.

Im Falle von Axis Aligned Bounding Boxen sind die beiden relevanten Achsen die x und y Achse des Koordinatensystems, da diese parallel zu den Normalen Körper liegen. Sobald eine Überschneidung festgestellt wird (auf der x oder y Achse) kann abgebrochen werden, da sich die Körper überschneiden.

3 Space Partitioning

Durch das Verwenden von Bounding Volumes wurden zwar die Kosten eines einzelnen Test stark reduziert, allerdings ist es immernoch notwendig alle Objektpaare zu testen. Dieses Problem wird durch Space Partitioning gelöst. Die Idee dahinter ist, dass ein Kollisionstest zwischen zwei Objekten der Spielwelt, die jeweils an einem Ende einer virtuellen Stadt sind, niemals positiv ausgehen kann. Allgemeiner gesagt: Zwei Objekte in verschiedenen Regionen der virtuellen Welt können sich nicht überschneiden und müssen deswegen auchnicht getestet werden. Aufbauend auf dieser Erkenntnis werden im Folgenden einige Datenstrukturen eingeführt mit Hilfe derer solche Partitionierungen der Spielwelt effizient implementiert werden können.

3.1 Regular Grids

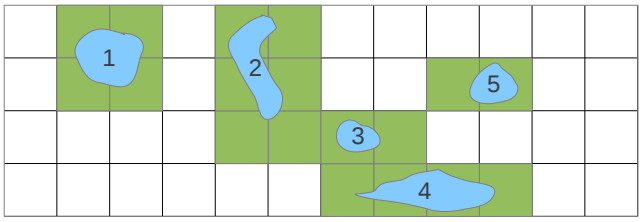

Abbildung 5: Ein reguläres Gitter.

Bei Regulären Gittern wird wie in Abbildung 5 gezeigt ein Gitter über die gesamte Spielwelt gelegt. Die Größe der Gitterzellen ist überall gleich. Jede Gitterzelle speichert die Objekt, die sich in ihr befinden. Die vier in grün hervorgehobenen Gitterzellen unter $Objekt\ 1$ speichern in dem Beispiel einen Verweis auf $Objekt\ 1$. Alle Objekte mit denen sich $Objekt\ 1$ möglicherweise schneiden kann befinden sich in diesen vier Gitterzellen. In diesem Beispiel müsste also für $Objekt\ 1$ kein einziger Kollisionstest ausgeführt werden.

3.1.1 Gittergröße

Bei Regulären Gittern fällt der Wahl der Gittergröße eine wichtige Rolle zu. Wird die Gittergröße zu groß gewählt hat dies zur Folge, dass sich viele Objekte in einer Gitterzelle befinden, was wiederum in vielen Kollisionstests und damit schlechter Performance resultiert. Wenn, auf der anderen Seite, die Gittergröße zu gering gewählt wird verbraucht das Gitter sehr viel Speicherplatz. Außerdem wird das Bewegen der Objekte sehr teuer, da die Verweise in allen Gitterzellen, über die sich das Objekt erstreckt, überprüft werden müssen.

Die Gittergröße so zu wählen, dass das größte Objekt der Spielwelt in einer Zelle platz findet ist eine gute Faustregel. Dies erleichtert auch einige Berechnungen, da sich ein Objekt so maximal in vier Gitterzellen befinden kann.

Ein Problem entsteht allerdings wenn die virtualle Welt, die auf das Gitter abgebildet wird, aus Objekten von sehr unterschiedlicher Größe besteht. In diesem Fall sollte man die Verwendung eines Trees – wie in 3.2 beschrieben – oder eines H-Grids – wie in [3] beschrieben – in Erwägung ziehen.

Ein weiteres Problem, welches aus der nicht dynamischen Partitionierung des Raumes folgt, sind geclusterte Objekte. Oft befinden sich in Spielen – wie in zu Beispiel während einer Schlacht – viele Einheiten auf einem kleinen Bereich der Spielwelt und damit in einigen wenigen Gitterzellen. Dies resultiert in vielen Kollisionstests und damit in schlechterer Performance.

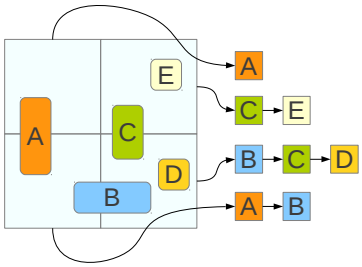

Abbildung 6: Ein reguläres Gitter.

3.1.2 Beispiel

Im Folgenden soll eine simple Implementierung eines Regulären Gitters vorgestellt werden (vgl. Abbildung 6). Jede Gitterzelle enthält eine List mit in ihr enthaltenen Objekten. In der Graphik ist auf der linken Seite das Gitter und auf der rechten Seite die Listen zu sehen. Die Pfeile zeigen an welche Liste zu welcher Gitterzelle gehört.

Um die möglichen Kollisionspartner von $Objekt\ C$ zu finden müsste man zuerst die Listen, die zu den Gitterzellen, in denen sich $Objekt\ C$ befindet, gehören, finden. Dies sind die beiden mittleren Listen mit dem gemeinsamen Inhalt: E, B, D, C. Das sind die Objekte mit denen $Objekt\ C$ auf eine Kollision getestet werden muss.

Für weitere Implementierungen sei auf [3] verwiesen.

3.2 Trees

Eine zweite Möglichkeit um die Spielwelt in disjunkte Bereiche zu partitionieren ist die Verwendung von Bäumen. Dieser Ansatz partitioniert die virtuelle Welt im Gegensatz zu den Regulären Gitter dynamisch. Dadurch sind Bäume inherent in der Lage geclusterte Daten und Objekte mit sehr unterschiedlichen Größen zu verwalten. Zu den klassischen Vertretern gehören hier die Binary Space Partitioning Trees (BSP-Tree), auf die im Folgenden eingegangen wird. Die auch weit verbreiteten k-dimensional Trees (kd-Trees) und die Quad-Trees, sind spezialisierte Binary Space Partitioning Trees, für deren Details auf [3] verwiesen sei.

3.2.1 Binary Space Partitioning Trees

Binary Space Partitioning Trees werden verwendet um eine zwei dimensionale Fläche in disjunkte Partitionen zu teilen. Hierzu wird eine Gerade durch die Fläche gelegt um zwei Teilflächen zu erzeugen. Auf diese Weise kann eine Fläche rekusiv immer feiner partitioniert werden. Ein Knoten im Baum stellt eine Unterteilung da, die Kinder des Knotens stellen die durch die Unterteilung erzeugten Teilflächen dar. In Abbildung 7 ist auf der linken Seite die virtuelle Welt zu sehen, die der Binary Space Partitioning Tree unterteilen soll, und auf der rechten Seite der Baum selbst.

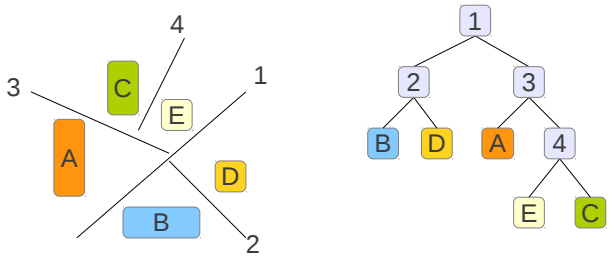

Abbildung 7: Beispiel eines BSP-Trees.

3.2.2 Eigenschaften von Bäumen

In diesem Abschnitt sollen die wichtisten Entscheidungen bei der Entwicklung eines effizienten Binary Space Partitioning Trees erläutert werden. Diese können analog auch auf die Entwicklung von kd-Trees und Quad-Trees übertragen werden.

- Wie in 3.2.1 beschrieben unterteilt der Baum die virtuelle Welt rekursiv in feinere Teilgebiete. Da die Unterteilung, aus offensichtlichen Gründen, nicht beliebig fein werden kann muss es eine Abbruchbedinung geben. Eine Möglichkeit wäre es die golbale Tiefe des Baumen zu limitieren oder bei einer bestimmten Anzahl an Objekten in einem Knoten abzubrechen.

- Ein weiterer Punkt ist die Wahl der Lage der Teilungsgeraden. Bei den Quad Trees ist die Lage der Geraden bestimmt. Dadurch muss kein Aufwand betrieben werden, um die Teilgebiete zu berechnen und es ist, durch die achsenorientiertheit der Geraden leichter ein Objekt zu finden. Auf der anderen Seite ist der Baum dadurch weniger dynamisch an die Daten angepasst und wird eventuell tiefer. Bei den Binary Space Partitioning Trees muss die Lage der Teilungsgeraden berechnet werden. Eine Strategie hierfür könnte so aussehen, dass in jedem Schritt die Position der Objekte der virtuellen Welt analysiert wird und die Teilungsgerade so gewält wird, dass auf beiden Seiten gleich viele Objekte liegen. Es muss also mehr Aufwand für das Berechnen und finden der Teilgebiete betrieben werden, dafür kann sich der Baum aber schneller anpassen. Die k-dimensionale Trees sind eine Art Mittelweg, da die Rotation der Geraden schon festgelegt ist und nur die Position geählt werden muss.

- Desweiteren sollte der Frage, wie ein Objekt behandelt wird, welches von einer Teilungsgeraden durchschnitten wird, Aufmerksamkeit geschenkt werden. Verbreitete Strategien liegen daran das fragliche Objekt in beiden Kinderknoten einzufügen oder direkt in den Parent-Node.

- Ein letzter Punkt soll dem Verzweigungsgrad der Bäume gewidtmet werden. In der Standard Implementierung ist der Verzweigungsgrad von Binary Space Partitioning Tress zwei – jeder Knoten hat zwei Kinder. Es ist aber durchaus möglich und auch sinnvoll dies zu erhöhen, da so die Tiefe des Baumes verringert werden

kann, wodurch die zu traversierenden Knoten bei einer Suche reduziert werden. Dies kann bei großen Bäumen die Anzahl an Cache-Misses reduzieren und so die Performance verbessern.

4 Conclusion

In dieser Ausarbeitung wurde eine Übersicht über die Broad Phase der Kollisionserkennung gegeben. Es wurden in Abschnitt 3 die verschiedene standard Techniken, wie Reguläre Gitter und Bäume, für das Space Partitioning gezeigt. Davor wurde der Sinn und die Anwendung von Bounding Volumes in Abschnitt 2 erklärt. Zusammen stellen diese Techniken eine gute Grundlage für ein effizientes System zur Kollisionserkennung dar. Die endgültige Wahl für ein konkretes Verfahren muss dem Anwender, der diese anhand der Situtation wählen muss, überlassen werden.

Literatur

[1] *Vorlesungs Folien aus Simulation and Animation – SS 2011 von K. Bürger, C. Dick.*

[2] William Bittle. *SAT (Separating Axis Theorem).* http://www.codezealot.org/archives/55.

[3] Christer Ericson. *Real-Time Collision Detection.* Morgan Kaufmann, 2005.