

2006

Kollisionserkennung mit der GPU

Dominik Riehl, Gunther Hohmann,

Steffen Höhmann

Universität Kassel

6/28/2006

Seminar, 6. Semester

bei Prof. Claudia Leopold,

Betreuung: Björn Knafla

Inhalt

ÜBERSICHT	2
MÖGLICHKEITEN DER GPU	3
INTERAKTIVE KOLLISIONSERKENNUNG	3
WICHTIGE KOLLISIONSERKENNUNGS-TECHNIKEN	4
ALGORITHMEN ZUR KOLLISIONSERKENNUNG AUF DER GPU	6
CULLIDE	6
QUICK-CULLIDE	9
GESAMTFAZIT	11
REFERENZEN	12

Übersicht

Unter Kollisionserkennung versteht man die Überprüfung, ob sich ein bestimmter Bereich in einem Raum mit einem anderen Bereich überschneidet. In Computerprogrammen handelt es sich dabei heutzutage meistens um einen 3-Dimensionalen Raum. Kollisionserkennung wird eingesetzt in Computerspielen, Simulationen und „Virtual Reality“ Anwendungen. Auch bei der Entwicklung von Robotern ist sie von Bedeutung. Auf den folgenden Seiten werden bisherige Kollisionserkennungstechniken vorgestellt, sowie die Vorteile sie auf einer Grafikkarte einzusetzen erläutert. Anschließend werden folgende GPU basierte Algorithmen ausführlich vorgestellt: CULLIDE, Quick-CULLIDE. Auf spezielle CPU Algorithmen wird nicht näher eingegangen.

CULLIDE und Quick-CULLIDE schaffen es den Zeitaufwand zur Kollisionsberechnung, durch Ausnutzung der Vorteile der GPU gegenüber der CPU, auf noch nie erreichte Werte zu reduzieren.

Möglichkeiten der GPU

Der folgende Abschnitt wird kurz dargestellt warum sich die Verwendung der GPU überhaupt für Kollisionsberechnungen empfiehlt.

GPUs eignen sich hervorragend um Sichtbarkeitsberechnungen durchzuführen. Darunter versteht man eine Technik, die durch jede Grafikkarte zur Verfügung gestellt wird. Bei dieser wird überprüft ob ein bestimmtes Objekt in einem 3-Dimensionalen Raum sichtbar ist. Dabei wird das Bild, indem sich eine Vielzahl von Objekten befindet, Schrittweise aufgebaut. Objekte werden beim Aufbau des Bildes anhand der Koordinatenachsen im Raum angeordnet. Aufgebaut sind Objekte durch ein Vielzahl von Dreiecken. Wird also ein Objekt in den Raum gezeichnet muss dabei jedes einzelne Dreieck gezeichnet werden. Von Vorteil sind hierbei die enormen parallelen Berechnungen, die eine GPU ermöglicht. Dadurch können Millionen von Dreiecken pro Sekunde gerendert werden.

Entscheidend für Verfahren der Kollisionserkennung ist die Genauigkeit mit der sie arbeiten. Je exakter die Berechnungen durchgeführt werden, desto länger dauern sie. Bei Algorithmen, die Bilddaten verwenden und ihre Berechnung durch die CPU durchführen lassen, wird die Laufzeit stark von den so genannten „readbacks“ beeinflusst. Bei diesen werden Inhalte der Grafikkarte zur CPU übertragen. Ein Framebuffer readback benötigt bei einer Buffer-Auflösung von 1000*1000 Pixeln selbst auf einer modernen PCI Express Grafikkarte noch knapp über 15ms. Werden die Berechnungen von der GPU übernommen müssen die Bilddaten nicht mehr zur CPU transferiert werden.

Interaktive Kollisionserkennung

Im Folgenden werden einige typische Schwierigkeiten vorgestellt, mit denen Kollisionsverfahren umgehen können sollten.

Ein Raum kann aus einer hohen Zahl von Objekten bestehen, die wiederum aus einer großen Menge aus Dreiecken bestehen können. Trotz dieser hohen Zahl sollten die Berechnungen in vielen Fällen noch in Echtzeit durchgeführt werden können. Objekte können offen oder geschlossen sein. Geschlossen bedeutet, dass die Dreiecke des Objekts eine geschlossene Fläche bilden müssen. Ein Objekt ohne geschlossene Oberfläche gibt Einblick in das texturlose Innere. Die Kollisionserkennung muss sich auch auf Bewegungen, zeitliche Änderungen der Objekte, einstellen. Objekte können sich bewegen, verformen oder ihre Oberfläche verändern. Dabei kann es auch passieren, dass es zu einer Kollision eines Objekts mit sich selbst kommt.

Wichtige Kollisionserkennungs-Techniken

Es gibt zwei entscheidende Techniken für die Kollisionserkennung. Dies sind die „Objekt-Raum“ Techniken und die „Bild-Raum“ Techniken.

OBJEKT-RAUM KOLLISIONSTECHNIK¹

Die „Objekt-Raum“ Techniken werden umgesetzt durch die sogenannte 2-Phasen-Kollisionserkennung. Ziel dieser Technik ist es möglichst effizient Kollisionen von sehr vielen Objekten festzustellen. Normalerweise müsste jedes Objekt mit jedem Anderem auf eine mögliche Kollision überprüft werden. Dies hätte einen Aufwand von $O(n * n) = O(n^2)$ zur Folge. Um die gewünschte Effizienz zu erreichen empfiehlt es sich durch geeignete Vorberechnungen die Anzahl der zu überprüfenden Paare zu reduzieren.

Wie diese Vorberechnungen erreichen können, soll folgendes kleines Beispiel verdeutlichen. Grundlage dabei ist eine Billardsimulation. Auf dem Tisch sollen alle Kugeln auf Kollisionen überprüft werden. Dazu kann ein ganz simpler Algorithmus verwendet werden, der einfach nur die Seite des Tisches bestimmt auf dem jede Kugel liegt. So brauchen für die anschließende Kollisionserkennung für jede Kugel nur noch die Kugeln beachtet werden, die sich auf ihrer Seite befinden. Liegen also z.B. Kugel 1-3 auf einer Seite und Kugeln 4-15, sowie die Weiße, auf der Anderen, dann muss für Kugel 1 nur eine Kollisionsüberprüfung mit den Kugeln 2 und 3 durchgeführt werden. Anschließend muss dies für die restlichen Kugeln entsprechend durchgeführt werden. Damit können sich zwar im worst case alle Kugeln auf einer Seite befinden, im Durchschnitt ist dies aber immer noch effizienter als ohne Vorbereitung.

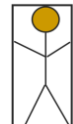
¹ Stefan M. Grünvogel, Kollisionserkennung bei 3D Objekten

2-PHASEN-KOLLISIONSERKENNUNG

Die 2-Phasen-Kollisionserkennung beginnt mit der sogenannten Broad-Phase (globale Phase). Dabei werden alle eng beieinander liegende Objekte bestimmt. So können alle Objekte, die eindeutig nicht für eine Kollision miteinander in Frage kommen, für die weiteren Berechnungen ausgeschlossen werden. Um diese Phase nicht zu rechenintensiv zu gestalten werden Bounding Volumes²(BV) um die Objekte gelegt. So müssen zur Abstandsberechnung zweier Objekte nur noch die Eckkoordinaten (Mittelpunkt+Radius bei der Kugel) der BV miteinander verglichen werden, anstatt die Abstände aller Dreiecke zu bestimmen. Es gibt verschieden Arten von BV. Bei diesen ist es entscheidend wie groß das umschlossene Volumen ist, abzüglich des Volumens des umschlossenen Objekts. Je größer das Volumen umso ungenauer ist die Kollisionserkennung. Um ein möglichst kleines Volumen zu erreichen, sind komplexere BV nötig, die sich eng um ein Objekt legen. Mit steigender Komplexität steigt aber auch der Rechenaufwand zur Erstellung der BV und der Abstandsberechnungen. Die gängigsten BV werden im folgenden kurz erläutert. Die Reihenfolge spiegelt dabei die Komplexität der Abstandsberechnung wieder und dem Grad wie eng Objekte umschlossen werden. Begonnen wird mit der einfachsten Technik und dem größtem Volumen.

Bounding Volumes

- ✚ Sphere: Hierbei handelt es sich um eine Kugel, deren Mittelpunkt so gewählt wird, dass das gesamte Objekt umschlossen wird und der Radius minimal ist.
- ✚ AABB (axis-aligned bounding box): Ein Quader um das Objekt, dessen Linien parallel zu den Achsen des Raumes laufen. Der Quader ist dabei möglichst klein.
- ✚ OBB (oriented bounding box): Wie AABB, nur orientieren sich die Linien diesmal nicht an den Achsen des Raumes, sondern am Objekt. So kann das Volumen für gewöhnlich verringert werden.
- ✚ n-dop: n-dop beginnt meistens wie AABB. Anschließend wird das Volumen aber noch durch weitere Begrenzungsflächen verringert. Die Anzahl der Flächen wird durch das n angegeben. Die Flächen werden so gewählt, dass sich das Volumen möglichst stark verringert.
- ✚ Konvexe Hülle: Eine Hülle, die sehr eng am Objekt liegt. Dadurch werden allerdings auch viele Begrenzungsflächen benötigt. Bei einer konvexen Hülle befinden sich alle Linien, die man zwischen 2 beliebigen Punkten der Hülle ziehen könnte, innerhalb der Hülle.



² Stefan M. Grünvogel, Kollisionserkennung bei 3D Objekten

Nachdem nun nur noch Objekte die sehr wahrscheinlich in Kollisionen verwickelt sind übrig bleiben, beginnt Phase 2. Diese wird als Narrow Phase (lokale Phase) bezeichnet. In dieser Phase werden die verbliebenen Objekte auf exakte Kollision überprüft. Dies geschieht auf Ebene der Dreiecke. Dadurch ist es möglich genau festzustellen, welches Dreieck sich mit welchen Anderen überschneiden hat. Ein Algorithmus, der solche Berechnungen ermöglicht, ist z.B. der Lin-Canny-Algorithmus.

Die 2-Phasen Kollisionserkennung schafft es die Anzahl der zu vergleichenden Objekte zu verringern, indem vor der exakten Kollisionsbestimmung zusätzliche Vorberechnungen durchgeführt werden. Diese führen zwar insgesamt zu effizienten Berechnungen, sind von ihrem Aufwand her aber nicht zu vernachlässigen. An dieser Stelle setzen die Bild-Raum Techniken an. Diese versuchen wesentliche schnellere Vorberechnungen durchzuführen und dabei noch mehr nicht kollidierende Paare frühzeitig auszuschließen.

BILD-RAUM TECHNIKEN³

Die „Bild-Raum“ Techniken verwenden im Gegensatz zu den „Objekt-Raum“ Techniken nicht nur die CPU, sondern auch die Grafikkarte. Die GPU wird nicht grundsätzlich eingesetzt, sondern findet erst Einsatz in den Algorithmen, die im nächsten Abschnitt vorgestellt werden. Durch Zugriff auf die Grafikkarte, bzw. deren Buffer, ist es möglich das Bild zu betrachten, wie es bei der Ausgabe am Bildschirm aussehen würde. Dabei betrachten die Algorithmen das aktuelle Bild um zu entscheiden an welchen Stellen Kollisionen vorliegen. Diese Verfahren bringen aber einige Nachteile mit sich. Es kann nur mit geschlossenen Objekten gearbeitet werden. Ein Arbeiten mit einer willkürlichen Ansammlung an Dreiecken ist also nicht möglich. Außerdem sind sie sehr abhängig von der Auflösung. Je geringer die Auflösung, desto ungenauer wird das Ergebnis. Der entscheidende Geschwindigkeitsnachteil liegt in den bereits erwähnten Framebuffer readbacks. Der Einsatz der Bild-Raum Techniken auf der CPU scheint also nicht wirklich sinnvoll. Die Prinzipien dieser Technik bilden allerdings den Grundstock für die folgenden Algorithmen, deren Ziel es ist, all diese Probleme zu beheben.

Algorithmen zur Kollisionserkennung auf der GPU

Im Folgenden werden zwei Verfahren zur Kollisionserkennung unter Verwendung der Grafikkarte vorgestellt. Beide basieren auf dem gleichen Grundprinzip, dem von CULLIDE.

³ Stefan M. Grünvogel, Kollisionserkennung bei 3D Objekten

CULLIDE basiert auf den Bild-Raum Techniken und ist der erste Algorithmus, der auf der GPU abläuft. Dadurch werden enorme Geschwindigkeitssteigerungen gegenüber den bisherigen Algorithmen erzielt. Im Anschluss wird Quick-CULLIDE vorgestellt, welches CULLIDE weiter optimiert und erweitert.

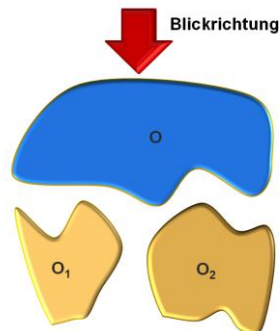
CULLIDE⁴

Der Kollisionserkennungsalgorithmus von CULLIDE läuft in drei Phasen ab. In der ersten Phase werden Berechnungen auf Objektebene durchgeführt um für Kollisionstests nicht relevante Objekte auszuschließen. In Phase zwei geschieht dies noch einmal auf Subobjekt Ebene. Als Subobjekt werden Objekte bezeichnet, die in mehrere einzelne Teile zerteilt sind. Diese beiden Phasen werden von der GPU durchgeführt. In Phase drei übernimmt die CPU und berechnet mit einem herkömmlichen Algorithmus die exakten Kollisionspunkte.

CULLIDE verwendet ein sogenanntes Potential Colliding Set (PCS). In diesem befinden sich alle Objekte die möglicherweise miteinander kollidieren. Zu Beginn befinden sich alle Objekte im PCS.

OBJEKT PHASE

In Phase 1 wird eine Sichtbarkeitsberechnung durchgeführt. Das folgende Bild soll veranschaulichen was man sich unter Sichtbarkeit eines Objekts vorstellen muss.



Das Bild wird aus der Sicht der eingezeichneten Perspektive betrachtet. Das vordere Objekt O wird bei einem Sichtbarkeitstest als vollständig sichtbar erkannt. Um dies festzustellen verwendet der Test den z-buffer. In diesem werden die Tiefeninformationen des Bildes gespeichert. Darin wird der z-Wert (Tiefenwert) jedes sichtbaren Pixels eines Objekts an der jeweiligen x-y Position eingetragen. Angenommen O₁ und O₂ werden vor O gezeichnet, dann befinden sich also die z-Werte ihrer Vorderseite, aus Sicht der Perspektive, im z-buffer. Wird nun der z-Wert von jedem Pixel von O mit den eingetragenen Werten verglichen, stellt der

⁴ Naga, G., Redon, S., Lin, M., & Manocha, D. (2003). CULLIDE

Sichtbarkeitstest fest, dass sie alle kleiner sind. Damit gilt O als vollständig sichtbar. Wäre nur ein Pixel größer oder gleich groß der bisher gespeicherten, schlägt der Test fehl. Die folgende Regel soll nun den Bezug zwischen Sichtbarkeit und Kollisionen herstellen.

Ein Objekt kollidiert nicht mit einer Menge anderer Objekte, wenn das Objekt in Bezug zu dieser Menge vollständig sichtbar ist.

Anhand dieser ersten Regel kann nun festgelegt werden, wann sich ein Objekt im PCS befindet.

Gegeben ist eine Menge n an Objekten, die in der Reihenfolge $O_1, O_2, \dots, O_i, \dots, O_n$ gezeichnet werden sollen. Ein Objekt O_i befindet sich dann nicht im PCS, wenn es nicht mit $O_1, O_2, \dots, O_{i-1}, O_{i+1}, \dots, O_n$ kollidiert.

Dieser Ansatz würde allerdings bedeuten, dass jedes Objekt mit allen anderen Objekten verglichen werden müsste, was einer Laufzeit von $O(n^2)$ entsprechen würde. Deshalb verwendet CULLIDE in der Objekt- und Subobjektphase ein 2-Wege-Render Verfahren.

FIRST PASS

Im ersten Schritt des 2-Wege-Render Verfahrens werden alle Objekte in der Reihenfolge gezeichnet, in der sie der Grafikkarte übergeben werden. Unter gezeichnet muss man in diesem Zusammenhang eine Rasterisierung des Objekts, also sozusagen eine Umwandlung in Pixel, verstehen. Die dabei erzeugten Bilddaten werden nur in den Grafikbuffern abgelegt um damit die Kollisionserkennung zu ermöglichen. Zur Anzeige werden sie nicht verwendet. Wird nun also ein Objekt O_i gezeichnet, befindet sich das Bild, wie es bis zu diesem Zeitpunkt aussehen würde, in den Buffern. Die Veränderungen die ein Objekt in den Buffern bewirken würde, werden nicht sofort gespeichert. Zuerst werden die Buffer gesperrt, O_i gezeichnet und darauf ein Sichtbarkeitstest ausgeführt. Scheitert der Test, wird O_i von einem anderen Objekt verdeckt oder berührt dies. In diesem Fall könnte eine Kollision vorliegen und O_i wird markiert. Die Markierung ist eine Informationsspeicherung zu jedem Objekt um festzustellen, ob es in beiden Schritten sichtbar ist. Das Objekt, welches O_i verdeckt, kann allerdings in diesem Schritt nicht markiert werden, da keine Informationen mehr über dieses Objekt verfügbar sind. Es wäre ein viel zu großer Aufwand all diese extra Informationen zu speichern und auch nicht notwendig, wie gleich zu sehen ist. Nachdem O_i nun eventuell markiert wurde, wird es noch einmal gezeichnet. Diesmal werden die Buffer aber freigegeben und die Veränderungen werden gespeichert.

SECOND PASS

Der Second Pass verläuft nach dem gleichen Prinzip wie der First Pass, nur werden die Objekte diesmal in umgekehrter Reihenfolge gezeichnet. So kann nun auch das Objekt

markiert werden, welches im 1. Durchgang nicht markiert wurde und O_i verdeckt hat. Da O_i nun vor den Objekten $O_1 \dots O_{i-1}$ gezeichnet wird, können für diese Veränderungen des z-Buffer mit Einfluss von O_i festgestellt werden. Ein Objekt wird dann aus dem PCS entfernt, wenn es in keinem der beiden Schritte markiert wurde.

SUBOBJEKT PHASE

Phase 2, die Subobjekt Phase, unterscheidet sich kaum von Phase 1. Ziel hierbei ist es wieder möglichst viele Objekte, bzw. Subobjekte aus dem PCS zu entfernen. Die Objekte werden in einzelne Subobjekte unterteilt. Auf welche Art und Weise dies geschieht ist absolut beliebig. Die Objekte können einfach in mehrere Quader eingeteilt werden oder sie können bis auf einzelne Dreiecke unterteilt werden. Diese Subobjekte werden genau wie in Phase 1 wieder auf mögliche Kollisionen überprüft und gegebenenfalls aus dem PCS entfernt.

CPU PHASE

In Phase 3 werden die verbleibenden Subobjekte an die CPU übergeben, welche dann mit einem Objekt-Raum Verfahren eine exakte Kollisionserkennung auf Dreiecksebene durchführt.

FAZIT ZU CULLIDE

Der Hauptrechenaufwand wird durch CULLIDE auf die Grafikkarte verlagert. Die CPU kann nebenbei andere Berechnungen durchführen. Die Operationen, die auf der Grafikkarte ausgeführt werden, sind für diese Standardoperationen. Dadurch ist der Aufwand minimal. CULLIDE versucht genau wie gute „Objekt-Raum“ Algorithmen möglichst viele Objekte als irrelevant zu markieren um so den endgültigen Rechenaufwand zu reduzieren. CULLIDE benötigt dazu aber nicht die erheblichen Vorberechnungen, die die CPU betreibt. Im Vergleich zu „Bild-Raum“ Algorithmen erspart sich CULLIDE die langsamen Readbacks. Dadurch, dass die Berechnungen von der GPU ausgeführt werden, müssen die Bufferdaten nicht immer wieder zur CPU transportiert werden. CULLIDE hat allerdings auch Nachteile, die im Folgenden zu sehen sind.

Quick-CULLIDE⁵

Quick CULLIDE enthält, wie der Name schon vermuten lässt, einiges an Optimierung, aber auch zusätzliche Funktionen gegenüber CULLIDE. Das 2-Wege Render Verfahren wurde verbessert. Außerdem werden Sichtbarkeitsbeziehungen aus verschiedenen Blickwinkeln verwendet. Dies ermöglicht es das PCS noch weiter zu reduzieren, wodurch der CPU noch mehr Arbeit abgenommen wird.

⁵ Govindaraju, N., Lin, M., & Manocha, D. (2005). Quick-CULLIDE: Efficient Inter- and Intra-Object Collision Culling using Graphics Hardware

Zu den Erweiterungen gehört auch eine Veränderung des PCS. Es gibt nun zusätzlich vier weitere PCS:

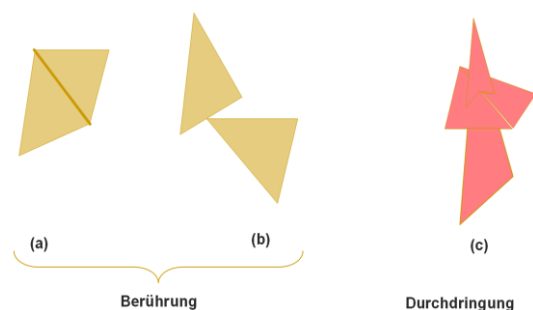
- ✚ FFV (First pass Fully Visible) speichert alle Objekte, die im ersten Durchgang als sichtbar erkannt werden.
- ✚ SFV (Second pass Fully Visible) speichert alle Objekte, die im zweiten Durchgang als sichtbar erkannt werden.
- ✚ NFV (Not Fully Visible in either passes) speichert alle Objekte, die in keinem Durchgang als sichtbar erkannt werden.
- ✚ BFV (Both passes Fully Visible) speichert alle Objekte, die in beiden Durchgängen als sichtbar erkannt werden.

Dabei gilt folgende Regel: **FFV und SFV sind Kollisionsfrei.**

Dies wirkt sich folgendermaßen auf das 2-Wege Render Verfahren aus: Zu Beginn befinden sich alle Objekte im NFV. Beim ersten Render Durchgang, dem aus CULLIDE bekannten First Pass, werden alle Objekte nach FFV verschoben, die als sichtbar erkannt werden. Beim Rendern in umgekehrter Reihenfolge, dem Second Pass, werden alle als sichtbar erkannten Objekte, die sich im NFV befinden, in den SFV verschoben, sowie alle die sich im FFV befinden in den BFV verschoben. Alle Objekte die sich jetzt im BFV befinden können aus dem PCS entfernt werden, da sie vollständig sichtbar sind und somit keine Kollision vorliegen kann. Dieses Verfahren bringt einen Geschwindigkeitsvorteil beim Rendern im Second Pass mit sich. Befindet sich nach dem Tiefentest ein Objekt im FFV oder BFV, muss es nun nicht mehr in die Grafikkbuffer gerendert werden. Denn wird ein Objekt im First Pass als vollständig sichtbar erkannt, können die nach diesem Objekt gerenderten Objekte im Second Pass nicht mehr mit diesem kollidieren. Dies wird durch die vollständige Sichtbarkeit im First Pass ausgeschlossen.

SELBSTKOLLISION

Quick-CULLIDE beherrscht im Gegensatz zu CULLIDE auch eine Technik zur Berechnung von Selbstkollisionen. Unter Selbstkollision versteht man es, wenn sich Dreiecke eines Objektes überschneiden. Dies tritt vor allem bei verformbaren Objekten wie Kleidung auf. Die Kollision der Dreiecke kann dabei entweder als



Berührung oder als Durchdringung auftreten. Zur Lösung dieses Problems müssen Berührungen ignoriert werden, da sie in jedem Objekt natürlicherweise vorhanden sind. Als Algorithmus wird wieder der CULLIDE Sichtbarkeitstest verwendet, welcher leicht angepasst

werden muss. Jedes Dreieck gilt nun als Objekt. Ein Objekt gilt als sichtbar und kann damit vom PCS entfernt werden, wenn folgende Regel gilt:

Ein Objekt ist vollständig sichtbar, wenn es nicht von einem anderem Objekt durchdrungen wird.

Im Tiefentest müssen die Objekte nun keinen Tiefenwert kleiner als die entsprechenden Pixel im z-Buffer haben, sondern es genügt, wenn sie einen Wert haben, der kleiner oder gleich groß ist. Dies ist ausreichend da Berührungen in diesem Fall nicht als Kollision gelten.

FAZIT ZU QUICK-CULLIDE

Quick CULLIDE erweitert CULLIDE auf sinnvolle Weise. In den meisten Fällen wird ein deutlicher Geschwindigkeitsvorteil erreicht und die Möglichkeit der Selbstkollisionserkennung wird gegeben.

Gesamtfazit

Die zwei vorgestellten Algorithmen CULLIDE und Quick-CULLIDE demonstrieren auf Eindrucksvolle Art und Weise, wie die Grafikkarte für die Kollisionserkennung ausgenutzt werden kann und dabei die bisher üblichen Probleme von Bild-Raum Techniken umgeht. Damit lässt sich unabhängig von Objektformen oder Bewegungen für eine große Anzahl an Objekten in kürzester Zeit eine Kollisionserkennung durchführen.

CULLIDE stellt eine gute Grundlage dar, die durch Quick-CULLIDE sinnvoll ergänzt und optimiert wird. Bei (Quick)CULLIDE gehen leider viele Kollisionen verloren, aufgrund von Fehlern bei der Bildabtastung und Ungenauigkeit des z-Buffer. Dafür wurde mit FAR⁶ noch ein dritter Algorithmus entwickelt, der diese Fehler beseitigt, dadurch aber auch wieder große Geschwindigkeitseinbuße hat. Quick-CULLIDE, welches nach FAR entwickelt wurde, greift deshalb die Ideen von diesem nicht wieder auf. Damit ist für alle Applikationen, die es etwas genauer möchten (Quick)CULLIDE nicht wirklich interessant. Hier müssten die Algorithmen noch um einiges verbessert werden.

Durch schlechte Blickwinkel und eine ungünstige Zeichenreihenfolge können ganze Objektberührungen verloren gehen. Hierzu ist es nötig (Quick)CULLIDE mehrfach aus unterschiedlichen Blickwinkeln auszuführen. Aber auch so kann nicht garantiert werden, dass wirklich alle Objekte erfasst werden. Dies führt in einer Szene mit z.B. 10000 Objekten

⁶ Govindaraju, N., Lin, M., & Manocha, D. (2004). Fast Collision Culling using Graphics Processors.

dazu, dass egal wie sie betrachtet wird, praktisch kaum ein Objekt aus dem PCS entfernt werden kann. Damit bleibt für die folgende CPU Berechnung fast der maximale Aufwand.

Entscheidend für viele Anwendungen, wie z.B. Computerspiele, dürfte auch die fehlende Intensität der Kollision sein. So benötigen viele Spiele Werte wie Aufprallgeschwindigkeit, Eindringungstiefe und Winkel. Einige CPU Algorithmen bieten diese Informationen, indem sie Zustände und zeitliche Veränderungen speichern. (Quick)CULLIDE betrachtet immer nur ein stehendes Bild und kann daher keine Aussagen über Bewegungen treffen. Für einige Anwendungen sind auch Abschätzungen über die Zukunft interessant. Diese werden anhand aktueller Bewegungsinformationen getroffen, was für (Quick)CULLIDE ebenfalls nicht möglich ist. Hier müsste noch ein besseres Zusammenspiel mit der verwendeten Objekt-Raum Technik entwickelt werden um dies zu ermöglichen.

Abschließend lässt sich sagen, dass (Quick)CULLIDE noch nicht vollständig ausgereift ist. Es wird aber schon deutlich gezeigt, dass für alle Anwendungen, die schnelle Kollisionsberechnungen benötigen, kein Weg an der GPU vorbeiführt.

Referenzen

Boldt, N., & Meyer, J. (2004). Self-intersections with Cullide. Retrieved from Eurographics 2004 CGF vol23 no3.

Govindaraju, N., Lin, M., & Manocha, D. (2004). Fast Collision Culling using Graphics Processors. Retrieved from University of North Carolina at Chapel Hill: <http://gamma.cs.unc.edu/RCULLIDE/far.pdf>.

Govindaraju, N., Lin, M., & Manocha, D. (2005). Quick-CULLIDE: Efficient Inter- and Intra-Object Collision Culling using Graphics Hardware. Retrieved from Department of Computer Science, University of North Carolina at Chapel Hill: <http://gamma.cs.unc.edu/QCULLIDE/QCULLIDE.pdf>.

Grünvogel, S. M. (2003). Kollisionserkennung bei 3D Objekten.

Kim, Y. (n.d.). Collision Detection for Large Gaming Environments. Retrieved from Dept of Computer Sci and Eng, Ewha Womans University: <http://graphics.ewha.ac.kr/>.

Naga, G., Redon, S., Lin, M., & Manocha, D. (2003). CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware. Retrieved from Department of Computer Science, University of North Carolina at Chapel Hill, U.S.A.: <http://www.cs.unc.edu/~geom/CULLIDE/cullide.pdf>.

Rege, A. (2002). GDC2002 -- Occlusion (HP and NV Extensions). Retrieved from nVidia: <http://developer.nvidia.com/attach/6715>.