„Interaktive Kollisionsvisualisierung von großen dynamischen Datensätzen der digitalen Fabrik“

Dipl.-Inf. Fabian Scheer
Daimler AG
Fabian.Scheer@daimler.com

Mario Marschner
Daimler Protics GmbH
Mario.M.Marschner@daimler.com

Zusammenfassung

Effiziente Kollisionsberechnungen werden im Bereich der computergrafischen Echtzeitanwendungen für eine überwiegende Mehrheit von Applikationen benötigt, und reichen von der Visualisierung, Simulation, Animation, Modellierung, Virtual und Augmented Reality bis hin zu den Computerspielen. Speziell große Datenmengen von dynamischen oder deformierbaren Objekten beliebiger geometrischer Form stellen dabei immer noch die besonderen Herausforderungen für interaktive oder echtzeitfähige Berechnungen dar. Unter Ausnutzung der weiterentwickelten Fähigkeiten moderner Grafikhardware und der enormen Fortschritte im Bereich bildraumbasierter Berechnungsverfahren stellen wir daher in diesem Beitrag ein Verfahren vor, um Kollisionen von extrem großen dynamischen Datensätzen (>10.Mio Polygone) der digitalen Fabrikplanung zu berechnen.

Schlüsselwörter

Interaktive Kollisionserkennung, Massive Datenmengen, Dynamische Objekte, Digitale Fabrik, Visuelle Absicherungsmethoden

1 Einführung und Motivation

Kollisionserkennung bezeichnet den Prozess der Erkennung von Objektdurchdringungen [JH08] und erfordert zeitaufwändige Berechnungen, welche zumeist gesonderte Datenstrukturen benötigen, um die Berechnungen zu beschleunigen. Dementsprechend ergibt sich durch die Beschleunigung dieser Berechnungen ein gravierender Vorteil für viele Anwendungen.

Unter Ausnutzung der weiterentwickelten Fähigkeiten moderner Grafikhardware wurden in den letzten Jahren einige Ansätze vorgestellt, die interaktive oder echtzeitfähige Bildwiederholraten erreichen [GKW07, JH08], wobei zunehmend mehr Aspekte der zugrundeliegenden Daten hinsichtlich dynamischer Inhalte, betrachteter Datenmenge, verschiedener Objekttypen und sogar deformierbarer Inhalte miteinbezogen werden können. Dennoch kommt es zumeist nur zu einer vereinzelten Betrachtung der verschiedenen Aspekte, so dass für eine Kombination komplexer dynamischer Szenen mit sehr großen Datenmengen, die beliebige Objekttypen und deformierbare Geometrien in großen räumlichen Ausdehnung beinhalten, immer noch keine angemessenen Lösungen existieren. Speziell in der letzten Dekade wurden viele rechenaufwändige Verfahren, wie z.B. Radiosity Simulationen [NSW09, NW09] oder Ambient Occlusion [BSD08, SK10], durch Berechnungen oder Approximationen im Bildraum, dem sog. Image Space, enorm beschleunigt, wodurch sich die folgenden Vorteile ergeben:

- Die Berechnungen können weitestgehend unabhängig von der betrachteten Menge an Polygonen durchgeführt werden und benötigen zumeist keine aufwändigen Vorberechnungen
- Bildraum-basierte Ansätze eignen sich speziell zur Berechnung voll dynamischer Szenen und können auch deformierbare Objekte berücksichtigen
- Die Berechnungen können komplett auf der Grafikkarte, der sogenannten Graphics Processing Unit (GPU), durchgeführt werden

Aufgrund der aufgeführten Vorteile wird daher die Verwendung von bildraum-basierten Ansätzen attraktiv für die Berechnung von Kollisionen, speziell wenn sehr große und dynamische Datenmengen (>10 Millionen Polygone) betrachtet werden sollen, wie sie insbesondere in der digitalen Fabrikplanung zum Einsatz kommen. Daher stellen wir in dieser Arbeit eine Methode vor, um Kollisionen in komplexen dynamischen Szenen zu berechnen. Das Verfahren basiert auf dem Ansatz von [JH08], überwindet allerdings die in [JH08] beschriebene Einschränkung auf einfache Szenen (lediglich 8 Fragmente werden betrachtet) und präsentiert eine effiziente Handhabung dynamischer Szenen mit sehr großen Datenmengen. In diesem Beitrag wird die Performanz des Verfahrens detailliert bewertet und es werden wichtige Faktoren analysiert, welche die Präzision und den Berechnungsaufwand beeinflussen. Das Verfahren ist in der Lage sehr große dynamische und deformierbare Datenmengen mit mehr als 10 Millionen Polygonen und großen räumlichen Ausdehnungen in interaktiven Bildwiederholraten zu betrachten und wird komplett auf der GPU ausgeführt, ohne Vorberechnungsschritte oder Datenstrukturen im Hauptspeicher zu benötigen. Aufgrund dieser Vorteile ist die Anwendung des Ansatzes in einer Vielzahl von Anwendungen im Bereich der Computergrafik möglich und soll in dieser Arbeit speziell für visuelle Absicherungsuntersuchungen in industriellen Anwendungen der digitalen Fabrikplanung betrachtet werden.

2 Stand der Forschung

Traditionell werden Beschleunigungsdatenstrukturen zur Berechnung von Kollisionen verwendet, wie z.B. Bounding Volume Hierarchien (BVHs), Octrees, Quadtrees oder Inner Sphere Trees [WZ09], die in einem Vorverarbeitungsschritt erzeugt und zur Laufzeit im Speicher (RAM) gehalten werden. In dieser Arbeit wird stattdessen eine Datenstruktur verwendet die zur Laufzeit auf der GPU generiert wird und keiner aufwändigen Vorberechnungszeiten für große Datenmengen bedarf, so dass die Zeit zur produktiven Nutzung und somit zur Markteinführung von Produkten oder Produktionsanlagen nicht erhöht wird.

2.1 Erfassung der Tiefe von mehreren Fragmenten

Verdeckte Geometrieinformation in Form von Fragmenten wird auf der aktuellen Grafikhardware durch einen Tiefentest verworfen. Diese verdeckte Information kann jedoch dennoch erfasst werden, d.h. es können mehrere Fragmente pro Pixel gespeichert werden. Zu diesem Thema wurden speziell in der letzten Dekade einige Ansätze vorgestellt, die im Folgenden kurz beschrieben werden. Als Vorreiter der Forschung in diese Richtung stellt [Car84] den A-Buffer vor, der eine Liste aller Fragmente pro Pixel speichert, die ferner entsprechend ihrer Tiefenanordnung sortiert sind. [Eve01] präsentiert Depth Peeling zur Erstellung der Fragmentliste, wobei ein separater Zeichenschritt der Geometrie (Render-Pass) für jede Tiefenebene der Szene durchgeführt werden muss. Dies kann von vorne nach hinten oder umgekehrt durchgeführt werden, so dass eine korrekt sortierte Liste aller Fragmente der Szene entsteht. Für komplexe Szenen mit großen Datenmengen ergibt sich allerdings ein enormer Rechenaufwand, da N Render-Passes benötigt werden, um N Tiefenebenen zu erfassen, so dass das Verfahren für die digitale Fabrikplanung als ungeeignet erscheint. [LBQ06] erfasst mehrere Tiefenebenen simultan durch Erweiterung des Depth Peeling Verfahrens, ist allerdings limitiert auf eine Präzision von lediglich 8-Bit pro Farbkanal und erreicht im Vergleich zu Depth Peeling nur eine doppelte Bildwiederholrate. In [BCL+07] wird auf Basis der Arbeit von [CC05] der K-Buffer vorgestellt, der simultan mehrere Fragmente durch Read-Modify-Write (RMW) Operationen erfassen kann. In diesem Zusammenhang berichtet [LFEH09] jedoch von gravierenden Artefakten des K-Buffers aufgrund von inkonsistenten Zuständen durch die RMW Operationen. Die Artefakte können durch eine dynamische Vorsortierung minimiert werden [BCL+07], was sich allerdings in deutlichen Geschwindigkeitseinbrüchen niederschlägt. [LFEH09] präsentiert ein effizientes Depth Peeling durch Sortierung der Fragmente mit dem Bucket Sort Algorithmus. Der Ansatz ist schnell zu berechnen und kann bis zu 32 sortierte Fragmente in einem Render-Pass erfassen. Nichtsdestotrotz ist das Verfahren auf eine Normalverteilung der Fragmente in der Szene angewiesen. Szenen deren Fragmente nicht gleichverteilt sind können zwar durch adaptives Bucket Depth Peeling berücksichtigt werden, jedoch verliert der Ansatz dadurch seinen Geschwindigkeitsvorteil gegenüber dem K-Buffer. [MB07] führt den Stencil Routed A-Buffer ein und nutzt die Multisamplingfähigkeiten moderner Grafik-

hardware, um bis zu 8 Fragmente pro Pixel durch Subpixel Stencil Routing in einem Render-Pass zu erfassen. Mehr als 8 Fragmente pro Pixel können durch die OpenGL Erweiterung „Occlusion Query“ detektiert und in einem zusätzlichen Render-Pass erfasst werden, so dass bis zu 254 Fragmente pro Pixel gespeichert werden können. Für die in dieser Arbeit fokussierten digitalen Fabrikplanungsszenarien mit massiven und komplexen Datensätzen stellt dies eine ausreichende Größenordnung dar. Die Einschränkung auf 254 Fragmente erfolgt lediglich aus der limitierten Auflösung des Stencil-Buffers von 8-Bit auf aktueller Grafikhardware. Die Datenstruktur des Stencil Routed A-Buffers wird in jedem Frame auf der GPU neu erzeugt und ist somit unabhängig von im RAM gehaltenen Daten.

2.2 GPU basierte Kollisionserkennung

Forschungsaktivitäten der letzten Jahren haben immer mehr effiziente Kollisionserkennungsalgorithmen hervorgebracht, welche unter Ausnutzung der Rechenleistung der GPU interaktive oder echtzeitfähige Bildwiederholraten erreichen und dynamische oder deformierbare Objekte berücksichtigen können. [HTG03, HTG04] präsentieren Layered Depth Images (LDIs), um Objektdurchdringungen anhand boolscher Operationen zu berechnen. Der Ansatz ist limitiert auf geschlossene Objekte und benötigt mehrere Render-Passes und Read-Back Operationen[1], um ein LDI für jedes Objekt zu erstellen. In [GRLM03] wird der Read-Back Flaschenhals minimiert und es werden Potentially Colliding Sets (PCS) verwendet, um Kollisionen auf Basis von Visibility Queries auf der GPU zu detektieren. Selbstkollisionen können nicht detektiert werden und die lineare Komplexität der Berechnungen lässt das Verfahren als ungeeignet für Szenen mit massiven Datenmengen erscheinen. In [GKJ+05] erweitern die Autoren ihren Ansatz, sind allerdings durch notwendige Vorberechnungen eingeschränkt, so dass dynamische Inhalte und große Datenmengen immer noch nicht adäquat behandelt werden können. [HYJH06] berechnet PCSs und Kollisionen komplett auf der GPU, ist allerdings auf geschlossene Objekte beschränkt und die Performanz hängt maßgeblich von der Anzahl der Polygone in der Szene ab. In [JJH07] erweitern die Autoren den Ansatz und können offene, sowie geschlossene Objekte durch Minimierung von teuren Read-Back Operationen berücksichtigen. Das Verfahren verwendet Depth Peeling und ist dementsprechend ungeeignet für eine interaktive oder echtzeitfähige Berechnung von Kollisionen in komplexen und massiven Datensätzen. Ferner ist die Erkennung von Selbstkollisionen nicht möglich. [GKW07] nutzen die Fähigkeiten moderner GPUs große Szenen effizient rasterisieren zu können, um Schnittkanten von Objektdurchdringungen zu berechnen. Leider ist das Verfahren auf geschlossene Objekte beschränkt. [JH08] präsentiert schnelle Kollisionsberechnungen zwischen dynamischen Objekten beliebiger polygonaler Form anhand eines A-Buffers. PCS werden durch einen Tiefenvergleich zwischen den Fragmenten eines Pixels identifiziert und per Read-Back auf die CPU transferiert,

[1] Read-Back bezeichnet die teure Zurückleseoperation von Daten von der GPU auf die CPU

um den Kollisionstest hochgenau auf der CPU zu berechnen. Nichtsdestotrotz können nur 8 Fragmente pro Pixel betrachtet werden, so dass Kollisionen in komplexen und massiven Szenen mit mehr als 8 Fragmenten pro Pixel nicht erkannt werden können.

3 Erzeugung mehrerer A-Buffer

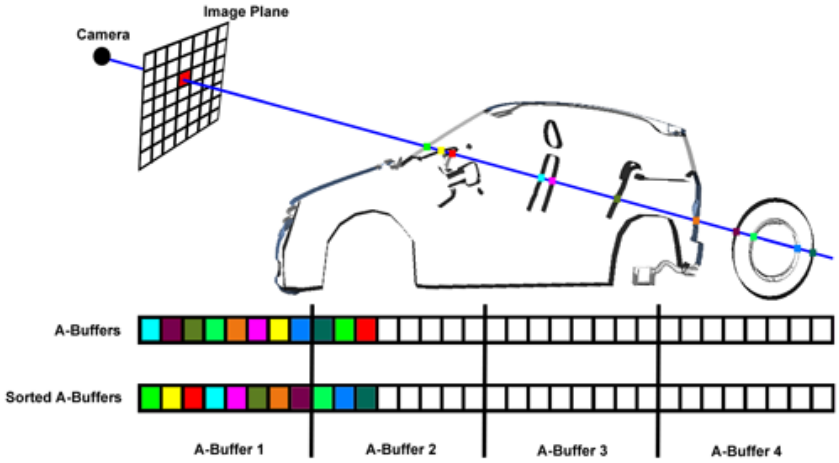

Abbildung 1: Befüllung mehrerer A-Buffer. Obere Reihe: Fragmente in der Reihenfolge ihrer Rasterisierung. Untere Reihe: Fragmente sortiert entsprechend ihrer Entfernung zur Bildebene. Leere Buffer werden zur Laufzeit nicht erzeugt.

Um verdeckte Fragmente innerhalb des aktuellen View Frustums zu erfassen werden in dieser Arbeit mehrere Stencil Routed A-Buffer verwendet. Es können 8 Fragmente mit einer Präzision von 16-Bit Fließkommawerten in einem A-Buffer gespeichert werden. Das Speichern in 32-Bit Genauigkeit ist auch möglich halbiert allerdings die Anzahl der in einem Render-Pass erfassbaren Fragmente auf 4. Da in massiven Szenen mehr als 8 Fragmente auf einen Pixel während der Rasterisierung fallen, werden mehrere A-Buffer verwendet, wobei ein Überlauf pro Pixel durch die OpenGL Occlusion Query detektiert und für jeden A-Buffer ein separater Render-Pass benötigt wird. Wir verwenden den Ansatz von [MB07], so dass maximal 254 Fragmente pro Pixel erfasst werden können. Erfahrungsgemäß langen im Mittel 8 A-Buffer zur Erfassung der Fragmente im View Frustum in den meisten Fabrikplanungsszenen, so dass wir uns auf die Erfassung von 64 Fragmenten pro Pixel in dieser Arbeit beschränken. Abbildung 1 verdeutlicht die Erstellung von zwei A-Buffern, um alle Fragmente der dargestellten Szene zu erfassen. Um komplexe und massive Datensätze berücksichtigen zu können, nutzen wir einen Visibility Guided Renderer (VGR) [KBHP07] zur Befüllung der A-Buffer. Zur effizienten und schellen Befüllung der A-Buffer profitieren wir maßgeblich vom Out-Of-Core Nachlademechanismus und den verschiedenen Culling Strategien des VGR, wobei der VGR so parametriert wird, so dass verdeckte Fragmente innerhalb des View Frustums im vorhinein nicht verworfen werden.

4 Präzision der Berechnungen

Die Fragmente in den A-Buffern werden mit dem 16-Bit Floating Point Datenformat erfasst. Dieses wird ”half precision“ genannt und belegt 2 Bytes im Speicher. Im IEEE 754-2008 Standard [IEE10] wird das 16-Bit zur Basis 2 Format offiziell mit binary16 bezeichnet und stellt ein relativ neues binäres Format zum Speichern von Fließkommazahlen dar. Die Kollisionsberechnung basiert auf diesem 16-Bit „half precision“ Datentyp. Diese auch Minifloats genannten Fließkommzahlen werden mit einem Bit für das Vorzeichen, 5 Bit für den Exponenten und einer 10 Bit breiten Mantisse gespeichert, wodurch Werte im Bereich von $\left[2^{-14}\ldots 65504\right]$ angegeben werden können. Somit ergibt sich ein minimal darstellbarer Wert von 0.000061 und ein maximal darstellbarer Wert von 65504, wobei auf die Konstante +INF gerundet wird, falls das Maximum überschritten wird. Die minimal darstellbare arithmetische Differenz zwischen zwei Fließkommazahlen, auch Unit in the Last Place (ULP) genannt, dient für Rechenwerke als Genauigkeitsmaß. Für das betrachtete „half precision“ 16-Bit Float Format ergibt sich eine ULP von $\log_{10}\left(2^{11}\right)$. Somit ergibt sich eine darstellbare Präzision von 3.31 Dezimalstellen, d.h. die dritte Nachkommastelle ist für die hier betrachteten Berechnungen noch als signifikant zu betrachten. Die in dieser Arbeit betrachteten 3D Modelle der digitalen Fabrikplanung sind ausschließlich in Metern modelliert. Aufgrund des Wertebereichs der Minifloats können daher Werte im Bereich von 0.061mm bis 65km betrachtet werden, was für die betrachteten Szenarien vollkommen ausreichend ist. Wird stattdessen die Erfassung von lediglich 4 Fragmenten in 32-Bit Präzision in den A-Buffern bevorzugt, so ist der darstellbare Wertebereich des binary32 Formats $\left[2^{-126}\ldots 3.4\cdot 10^{38}\right]$ und die siebte Nachkommastelle ist noch als signifikant zu betrachten.

5 Bildraum-basierte Kollisionserkennung

Die in dieser Arbeit vorgestellte Kollisionserkennung basiert auf der Repräsentation der betrachteten 3D Geometrien als Fragmente in einem Pixelshader Programm. Eine bildraum-basierte Kollision wird dementsprechend als die absolute Differenz zwischen den Tiefenwerten von zwei Fragmenten eines Pixels betrachtet, die unter einen gewissen Schwellwert ε fällt:

$$\left|Tiefe_{F1} - Tiefe_{F2}\right| < \varepsilon \qquad (1)$$

Diese Berechnung ist nicht korrekt im mathematischen Sinne der Bestimmung eines exakten Schnittpunkts, sondern stellt eher die Kollision zwischen flächigen Geometriefragmenten dar. Dennoch ist das Ergebnis hinreichend genau für eine Vielzahl von Applikationen, die auf interaktive oder echtzeitfähige Berechnungen angewiesen sind, wie z.B. Computerspiele oder visuelle Absicherungsuntersuchungen auf Basis großer Datenmengen. Des Weiteren bevorzugen die Fabrikplaner eine Visualisierung von Kollisionen als Schnittkanten, die somit recht einfach erreicht wird (siehe Abb.2).

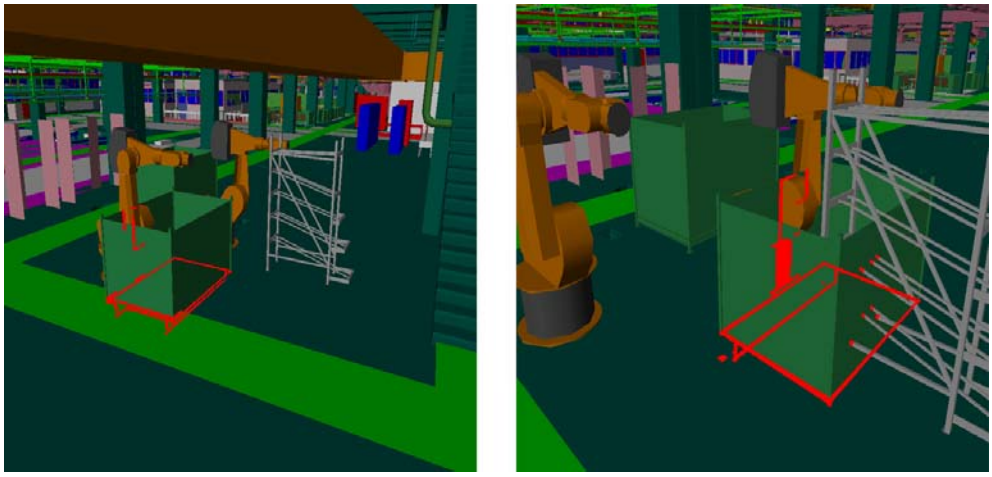

Abbildung 2: Kollisionsvisualisierung. Erkannte Kollisionen für die Fragmente eines Pixels werden durch Einfärbung des Pixels in rot als Schnittkanten visualisiert. Auch verdeckte Kollisionen werden grafisch als Schnittkanten dargestellt.

Aufgrund der in Kapitel 3 beschriebenen Erfassung von 64 Fragmenten pro Pixel können Szenen mit großen Datenmengen berücksichtigt werden. Gleichung 1 berechnet die Differenz zwischen den Tiefenwerten zweier Fragmente pro Pixel. Eine Brute Force Berücksichtigung zwischen allen Fragmenten n eines Pixels, würde daher zu $n!$ Vergleichen pro Pixel führen. Bei 64 Fragmenten und einer Bildauflösung von $1024x768$ Pixeln wäre der Rechenaufwand zu groß, um interaktive Bildwiederholraten zu erreichen. Werden die Fragmente hingegen anhand ihrer Tiefe sortiert, so muss die Differenz nur noch zwischen benachbarten Fragmenten durchgeführt werden (siehe Abb.1). [MB07] und [JH08] nutzen den Bitonic Sort Algorithmus mit 21 Vergleichen für eine Sortierung von 8 Fragmenten. Wir verwenden stattdessen den Odd-Even-Merge Sort Algortihmus, der nur 19 Vergleiche benötigt, um 8 Fragmente zu sortieren [Bat68]. Bei 64 Fragmenten ergeben sich 543 Vergleiche pro Pixel für 8 gefüllte A-Buffer. Wenn auch der Odd-Even-Merge Sort weniger Vergleiche benötigt, so erscheinen die Vergleiche für alle Pixel in Summe als zu kostspielige Operation. Eine weitere Optimierung besteht darin, potentiell kollidierbare Objekte anhand ihrer Objekt-ID zu identifizieren, so dass viele Fragmente ausgeschlossen werden können und deutlich weniger Texturzugriffe benötigt werden. Dazu muss allerdings die Objekt-ID eines Fragments im Shader bekannt sein. Zur Identifikation einzelner Objekte speichern wir die Objekt-ID als Shininess Parameter in den Objektmaterialien, so dass diese zur Kollisionsberechnung verwendet werden können. Für die finale Darstellung wird das ursprüngliche Material wieder hergestellt, so dass das Material im finalen Bild korrekt dargestellt wird. Betrachtet man nun einen Brute Force Vergleich nur für kollidierbare Objekte, so ergibt sich eine Summe von insgesamt

$$Pixel_{obj} \cdot \left(n_{obj} \cdot 8 \cdot N_{A-Buf} + \left(list_{frag} \cdot 8 \cdot N_{A-Buf}\right)\right)$$

Vergleichen, wobei n_{obj} die Anzahl kollidierbarer Objekte, $list_{frag}$ die Liste von einem kollidierbaren Objekt zugehörigen Fragmenten und N_{A-Buf} die Anzahl an A-Buffern bezeichnet. Für eine kleine Anzahl kollidierbarer Objekte (ca. 10-15) wäre dies äußerst effizient, allerdings steigt der Rechenaufwand proportional zur Anzahl der kollidierbaren Objekte. Stattdessen wird in dieser Arbeit eine alternative Methode verwendet, die zum Einen sehr viele kollidierbare Objekte berücksichtigt und zum Anderen von einer reduzierten Anzahl an Texturzugriffen profitiert. Dazu wird nur eine sortierte Liste der ersten sieben sich am nächsten zur Kamera befindlichen Fragmente erstellt. In diesem Zuge werden alle Fragmente in den A-Buffern gelesen und jedes Fragment das sich näher zur Kameraposition befindet als die sieben bereits gespeicherten Werte wird in einem separaten Array gespeichert. Die korrekte Einfügeposition unter den Fragmenten des ersten A-Buffers wird durch eine Binärsuche erledigt, wobei im schlimmsten Fall alle sieben Werte verschoben werden müssen. In dieser Arbeit wurde die Erfahrung gemacht, dass wenn der erste A-Buffer anhand des Odd-Even-Merge Sort Algorithmus vorsortiert wird, die Binärsuche beschleunigt wird. Durch Anwendung von Gleichung 1 kann somit eine Kollision zwischen den ersten 7 Fragmenten durch eine Differenzbildung benachbarter Fragmente berechnet werden, wobei auch Selbstkollisionen unter Einbeziehung der Objekt-ID detektiert werden können. Durch Beschränkung auf die ersten sieben Fragmente können interaktive Bildwiederholraten für komplexe Szenen mit massiven Datenmenge (>10 Millionen Polygone) erreicht und viele kollidierbare Objekte behandelt werden.

6 Ergebnis

In Tabelle 1 sind die Laufzeitmessungen der Kollisionserkennung für Szenen mit verschiedenen Datenmengen aufgeführt. Es wird zwischen der Berechnung der Kollision und der Erstellung der A-Buffer Datenstruktur unterschieden, so dass die Gesamtzeit in der Zeile „Summe“ aufgeführt wird. Für die Tests wurde ein Desktop PC mit einem Intel Core i7 920 Prozessor, 3GB RAM und einer NVIDIA GTX 285 Grafikkarte verwendet. Es konnten interaktive Bildwiederholraten für einfache und für komplexe Szenen mit großen Datenmengen erreicht werden (siehe Tabelle 1). Durch die Berechnung der Kollisionen anhand von Fragmenten können auch dynamische oder deformierbare Geometrien beliebiger Objekttypen ohne zusätzlichen Aufwand berücksichtigt werden, so dass im Vergleich zu anderen Ansätzen keine Einschränkungen in Form von geschlossen, konvexen oder konkaven Objekttypen oder die Einschränkung auf statische oder rigide Objekte gegeben sind. Die Erstellung der A-Buffer ist die teuerste Operation, die allerdings durch Verwendung kleinerer Auflösungen beschleunigt werden kann (siehe Tabelle 1).

Tabelle 1: Ergebnis der Laufzeitmessungen für die Kollisionserkennung in drei verschiedenen Testszenen in Millisekunden.

Szene Sibenik (3 Passes, 700.000 Polygone)			
A-Buffer Auflösung	200x200	400x400	800x800
Kollision	1.1 ms	3.6 ms	11.4 ms
A-Buffer Erzeugung	3 Passes * 1.6 ms	3 Passes * 1.8 ms	3 Passes * 3.3 ms
Summe	5.9 ms	9.0 ms	21.3 ms
Szene Transport (8 Passes, 3.2 Mio. Polygone)			
A-Buffer Auflösung	200x200	400x400	800x800
Kollision	2.6 ms	8.9 ms	30 ms
A-Buffer Erzeugung	8 Passes * 2.5 ms	8 Passes * 3.3 ms	8 Passes * 5.8 ms
Summe	22.6 ms	35.3 ms	76.4 ms
Szene Fabrik (8 Passes, 15.7 Mio. Polygone)			
A-Buffer Auflösung	200x200	400x400	800x800
Kollision	7.1 ms	22.1 ms	75.3 ms
A-Buffer Erzeugung	8 Passes * 11.7 ms	8 Passes * 12.2 ms	8 Passes * 15.5 ms
Summe	100.7 ms	119.7 ms	199.3 ms

Dazu werden die Berechnungen in der geringeren Auflösung durchgeführt und das Ergebnis im finalen Render-Pass für die Visualisierung durch ein Upsampling in der ursprünglichen Auflösung dargestellt. Sehr niedrig aufgelöste A-Buffer können sichtbare Artefakte verursachen (siehe Abb.3), sorgen allerdings dafür, dass das Verfahren auch auf leistungsschwächerer Hardware in interaktiven Bildwiederholraten berechnet werden kann. Abbildung 3 verdeutlicht dies exemplarisch. Die in Abbildung 3 gezeigte Szene besteht insgesamt aus 10.3 Millionen Polygonen, von denen sich ca. 8.2 Mio. im gezeigten Bildausschnitt befinden. Zur Erfassung aller Fragmente wurden 5 A-Buffer benötigt, die jeweils 65 ms zur Erstellung mit einer Auflösung von 400x400 Pixeln benötigen (Abb. 3, rechts). Der eingesetzte VGR Renderer gibt diesen Bildausschnitt in 13 Bildern pro Sekunde(fps) wieder. Durch Einschalten der Kollisionsberechnung auf der GPU sinkt diese Bildwiederholrate zwar auf 3 fps, dennoch werden für diese komplexe Szene mit ihren großen Datenmengen noch interaktive Bildwiederholraten erreicht. Abbildung 4 zeigt denselben Bildausschnitt mit mehreren kollidierenden dynamischen Objekten für den noch 2 fps erreicht wurden. Ein Nachteil bildraum-basierter Ansätze besteht darin, dass Flächen die parallel zur Kamerasichtrichtung verlaufen nicht rasterisiert werden und somit nicht in den A-Buffern erfasst werden können. Die Kollisionserkennung wurde jedoch primär für interaktive visuelle Absicherungsuntersuchungen der digitalen Fabrik (DIFA) entworfen, so dass ein geschulter Planer leicht mit dieser Tatsache durch Änderung der Blickrichtung umgehen kann. Darüber hinaus können Kollisionen außerhalb des aktuellen Bildausschnitts nicht berechnet werden. Durch die

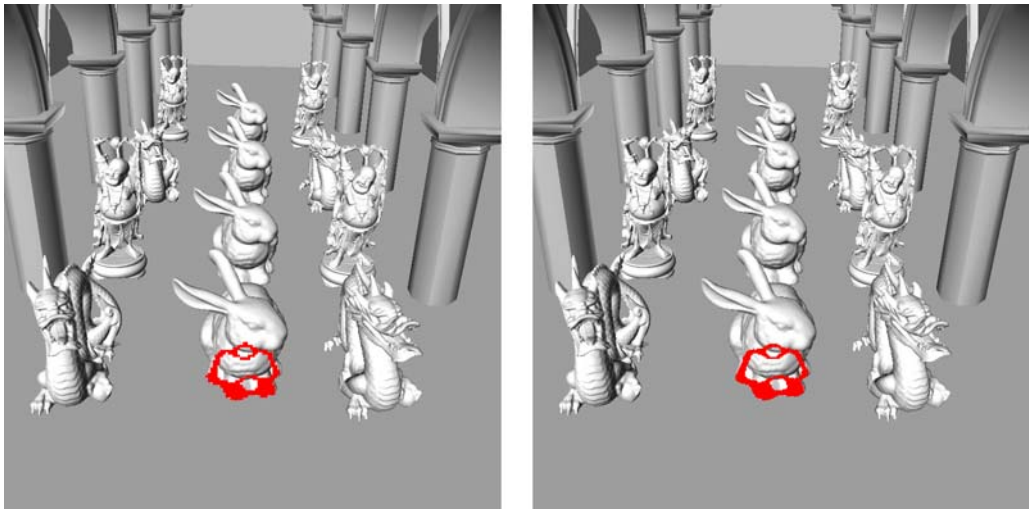

Abbildung 3: Kollisionserkennung auf A-Buffern mit einer Auflösung von 200x200 (links) und 400x400 Pixeln (rechts).

Fokussierung der Kollisionsberechnung auf den aktuellen Bildausschnitt können jedoch die Vorteile des Verfahrens, große dynamische und deformierbare Datenmengen beliebiger Objekttypen zu berücksichtigen, voll entfaltet werden. Zur Erreichung interaktiver Bildwiederholraten wird sich momentan auf Kollisionen zwischen den sieben am nächsten zur Kamera gelegenen Fragmente beschränkt, jedoch kann diese Grenze durch leistungsfähigere Grafikhardware in der Zukunft nach oben verschoben werden. Somit ist die Methode besonders für den Einsatz zur visuellen Absicherung von Planungsständen der DIFA auf Kollisionsfreiheit geeignet und ermöglicht Fabrikplanern die interaktive Überprüfung ihrer 3D Modelle. Letztendlich werden somit neue Möglichkeiten der digitalen Planung eröffnet, denn Änderungen von einzelnen Geometrien können schnell und einfach auf Kollisionsfreiheit überprüft werden, ohne zeitaufwändige Kollisionsberechnungen auf dem kompletten Datenstand durchführen zu müssen, die mitunter Stunden dauern können.

7 Resümee und Ausblick

In dem Beitrag wurde ein Verfahren vorgestellt, um Kollisionen auf Basis von bildraum-basierten Berechnungen in interaktiven Bildwiederholraten zu berechnen. Es können komplexe und massive Datensätze mit dynamischen und deformierbaren Inhalten von beliebig geformten Objekten berücksichtigt werden. Das Verfahren ist bestens geeignet, um von der Weiterentwicklung der Grafikhardware zu profitieren, so dass die Limitierung auf die ersten sieben Fragmente in Zukunft entfallen kann. Es sind keine Vorberechnungen oder gesonderte Datenstrukturen auf der Festplatte notwendig, wodurch sich enorme Vorteile für die digitale Fabrikplanung und auch für andere Anwendungsbereiche ergeben.

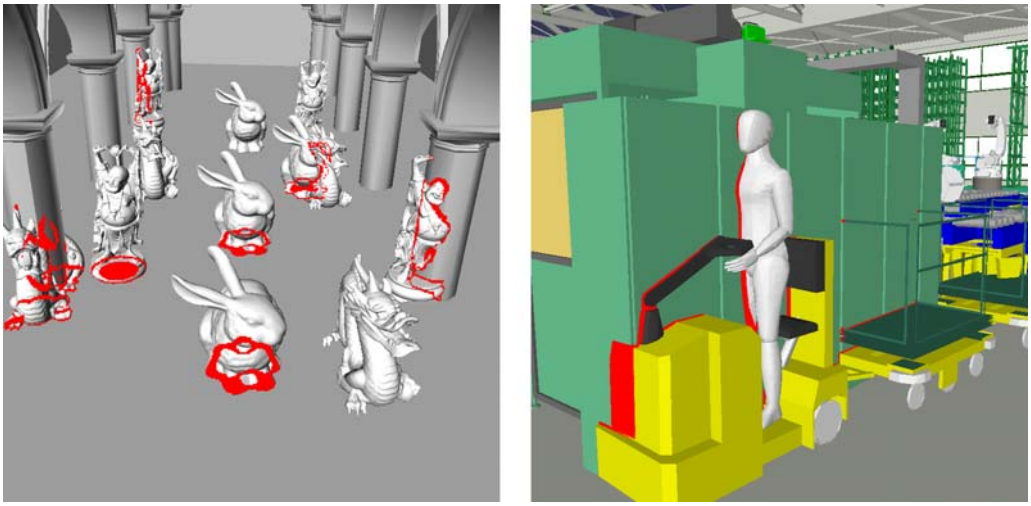

Abbildung 4: Kollisionserkennung mit großen Datenmengen auf A-Buffern mit einer Auflösung 400x400 für mehrere dynamische Objekte (links) und für eine typische Fabrikplanungsszene (rechts).

Literatur

[Bat68] KE Batcher. Sorting networks and their applications. In Proceedings of the spring joint computer conference, pages 307–314. ACM, 1968.

[BCL+07] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, ao L. D. Comba, Jo and Claudio T. Silva. Multi-fragment effects on the GPU using the k-buffer. In I3D ’07: Proceedings of the 2007 symposium on Interactive 3D graphics and games, pages 97–104, New York, NY, USA, 2007. ACM.

[BSD08] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In SIGGRAPH ’08: ACM SIGGRAPH 2008 talks, 2008. ACM.

[CC05] Steven P. Callahan and Joao L. D. Comba. Hardware-assisted visibility sorting for unstructured volume rendering. IEEE Transactions on Visualization and Computer Graphics, 11(3):285–295, 2005.

[Car84] L. Carpenter. The a-buffer an antialiased hidden surface method. In SIGGRAPH proceedings of the Conference on Computer graphics and interactive techniques, 1984.

[Eve01] C. Everitt. Interactive order-independent transparency. White paper, NVIDIA, 2(6):7, 2001.

[GKW07] Georgii, J., Krüger, J., Westermann, R.: Interactive GPU-Based Collision Detection. Proceedings of IADIS Computer Graphics and Visualization, 2007.

[GKJ+05] Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. ACM Trans. Graph., 24(3):991–999, 2005.

[GRLM03] N.K. Govindaraju, S. Redon, M.C. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pages 25–32. Eurographics Association Aire-la-Ville, Switzerland, 2003.

[HTG03] Bruno Heidelberger, Matthias Teschner, and Markus H. Gross. Real-time volumetric intersections of deforming objects. In VMV, pages 461–468, 2003.

[HTG04] Bruno Heidelberger, Matthias Teschner, and Markus H. Gross. Detection of collisions and self-collisions using image-space techniques. In WSCG, pages 145–152, 2004.

[HYJH06] TaekSang Jeong Han-Young Jang and JungHyun Han. Gpu-based image-space approach to collision detection among closed objects. In PG 2006: Proceedings of the 2006 Pacific Conference on Computer Graphics and Applications, pages 242–251, 2006.

[JH08] Jang, H., Han, J.: Fast Collision Detection Using the ABuffer.The Visual Computer: International Journal of Computer Graphics. 2008.

[JJH07] H.Y. Jang, T.S. Jeong, and J.H. Han. Image-space collision detection through alternate surface peeling. LECTURE NOTES IN COMPUTER SCIENCE, 2007.

[KBHP07] D. Kasik, B. Brüderlin, M. Heyer, and S. Pfützner. Visibility-guided rendering to accelerate 3d graphics hardware performance. In SIGGRAPH ’07: ACM SIGGRAPH 2007 courses, page 7, New York, NY, USA, 2007. ACM.

[LBQ06] Xu Y.-Q. Liu B.-Q., Wei L.-Y. Multi-layer depth peeling via fragment sort. Technical report, Microsoft Research Asia, 2006.

[LFEH09] Liu X.-H. Liu F., Huang M.-C. and Wu E.-H. Efficient depth peeling via bucket sort. In HPG ’09: Proceedings of the Conference on HighPerformance Graphics 2009, pages 51–57, New York, NY, USA, 2009. ACM.

[NSW09] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. Comput. Graph. Forum, 28(4):1141–1149, 2009.

[NW09] Greg Nichols and ChrisWyman. Multiresolution splatting for indirect illumination. In I3D ’09: Proceedings of the 2009 symposium on Interactive 3D graphics and games, pages 83–90, New York, NY, USA, 2009. ACM.

[MB07] Myers, K., Bavoil, L.: Stencil Routed A-Buffer. International Conference on computer Graphics and Interactive Techniques, 2007.

[SK10] Scheer, F., Keutel, M.: Screen Space Ambient Occlusion for Virtual and Mixed Reality Factory Planning. Journal of WSCG 18, 1-3, 2010.

[WZ09] Rene Weller and Gabriel Zachmann. Inner sphere trees for proximity and penetration queries. In 2009 Robotics: Science and Systems Conference (RSS), Seattle, WA, USA, June 2009.