HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

Titel der Arbeit auf Deutsch

English Title

Jeremias Boos

Hausarbeit zur Vorlesung Realtime Rendering

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, 7. März 2016

Kurzfassung

Dies ist eine eines Verfahrens zur Berechnung von 2D Schatten. Das Hier vorgestellte verfahren arbeitet hauptsächlich auf der GPU mit Framebuffern und Fragment Shadern.

Inhaltsverzeichnis

1

Motivation

Mich hat es schon immer interessiert wie Schatten in 2D Spielen umgesetzt. Dazu habe ich zwei verfahren gefunden. Das eine basiert darauf eine Physikengine zu nutzten und dann mit Hilfe von Raycasts einen “Lichtmesh“ zu erzeugen. Das andere Verfahren nutzt die Grafikkarte und berechnet die Schatten im Fragmentshader auf der GPU. Da sich der Prozess gut parallelisieren lässt und Grafikkarten immer stärker werden habe ich mich für die zweiter Variante entschieden.

2

Verfahren

Das Verfahren besteht grundsätzlich aus 3 Schritten.

2.1 Oclussionmap

Zu erst wir eine Oclussionmap gerendert. Diese stellt den Einflussbereich des Lichtes dar.2.1

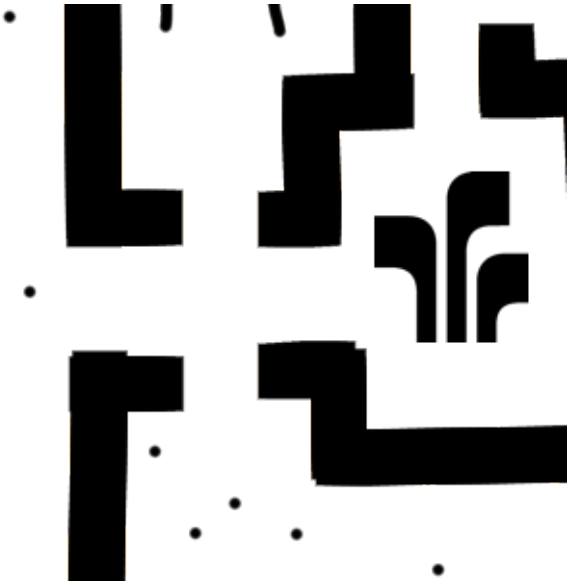

Abb. 2.1. Oclussionmap

2.2 Sample Distance

Anschließend wird das Oclusion Bild mit polarcoordinaten gesampled. Danach speichern wir die gesampelte minimal Distanz in einer 1D Textur. Als 1D Texture sieht das dann wie in Abbildung 2.4 aus.

Abb. 2.2. Oclussionmap in polar Koordinaten

Abb. 2.3. Symbolische Shadowmap in polar Koordinaten

Abb. 2.4. Gesampelte Distanzdaten.

3

Implementierung

4

Einleitung und Problemstellung

Begonnen werden soll mit einer Einleitung zum Thema, also Hintergrund und Ziel erläutert werden.

Weiterhin wird das vorliegende Problem diskutiert: Was ist zu lösen, warum ist es wichtig, dass man dieses Problem löst und welche Lösungsansätze gibt es bereits. Der Bezug auf vorhandene oder eben bisher fehlende Lösungen begründet auch die Intention und Bedeutung dieser Arbeit. Dies können allgemeine Gesichtspunkte sein: Man liefert einen Beitrag für ein generelles Problem oder man hat eine spezielle Systemumgebung oder ein spezielles Produkt (z.B. in einem Unternehmen), woraus sich dieses noch zu lösende Problem ergibt.

Im weiteren Verlauf wird die Problemstellung konkret dargestellt: Was ist spezifisch zu lösen? Welche Randbedingungen sind gegeben und was ist die Zielsetzung? Letztere soll das beschreiben, was man mit dieser Arbeit (mindestens) erreichen möchte.

5

Weitere Kapitel

Die Gliederung hängt natürlich vom Thema und von der Lösungsstrategie ab. Als nützliche Anhaltspunkte können die Entwicklungsstufen oder - schritte z.B. der Softwareentwicklung betrachtet werden. Nützliche Gesichtspunkte erhält und erkennt man, wenn man sich

- in die Rolle des Lesers oder
- in die Rolle des Entwicklers, der die Arbeit z.B. fortsetzen, ergänzen oder pflegen soll,

versetzt. In der Regel wird vorausgesetzt, dass die Leser einen fachlichen Hintergrund haben - z.B. Informatik studiert haben. D.h. nur in besonderen, abgesprochenen Fällen schreibt man in populärer Sprache, so dass auch Nicht-Fachleute die Ausarbeitung prinzipiell lesen und verstehen können.

Die äußere Gestaltung der Ausarbeitung hinsichtlich Abschnittformate, Abbildungen, mathematische Formeln usw. wird in Kapitel 6 kurz dargestellt.

6

LaTeX-Bausteine

Der Text wird in bis zu drei Ebenen gegliedert:

1. Kapitel (`\chapter{Kapitel}`),
2. Unterkapitel (`\section{Abschnitt}`) und
3. Unterunterkapitel (`\subsection{Unterabschnitte}`).

6.1 Abschnitt

Text der Gliederungsebene 2.

6.1.1 Unterabschnitt

Text der Gliederungsebene 3. Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Beispiel für Quelltext

```
1     Prozess 1:
2
3     Acquire();
4        a := 1;
5     Release();
6     ...
7     Acquire();
8     if(b == 0)
9     {
10       c := 3;
11       d := a;
12    }
13    Release();
```

```
1       Prozess 2:
2
3       Acquire();
4           b := 1;
5       Release();
6       ...
7       Acquire();
8       if(a == 0)
9       {
10          c := 5;
11          d := b;
12      }
13      Release();
```

Größere Code-Fragmente sollten im Anhang eingefügt werden.

6.2 Abbildungen und Tabellen

Abbildung und Tabellen werden zentriert eingefügt. Grundsätzlich sollen sie erst dann erscheinen, nach dem sie im Text angesprochen wurden (siehe Abb. 6.1). Abbildungen und Tabellen (siehe Tabelle 6.1) können im (fließenden) Text (`here`), am Seitenanfang (`top`), am Seitenende (`bottom`) oder auch gesammelt auf einer nachfolgenden Seite (`page`) oder auch ganz am Ende der Ausarbeitung erscheinen. Letzteres sollte man nur dann wählen, wenn die Bilder günstig zusammen zu betrachten sind und die Ausarbeitung nicht zu lang ist (< 20 Seiten).

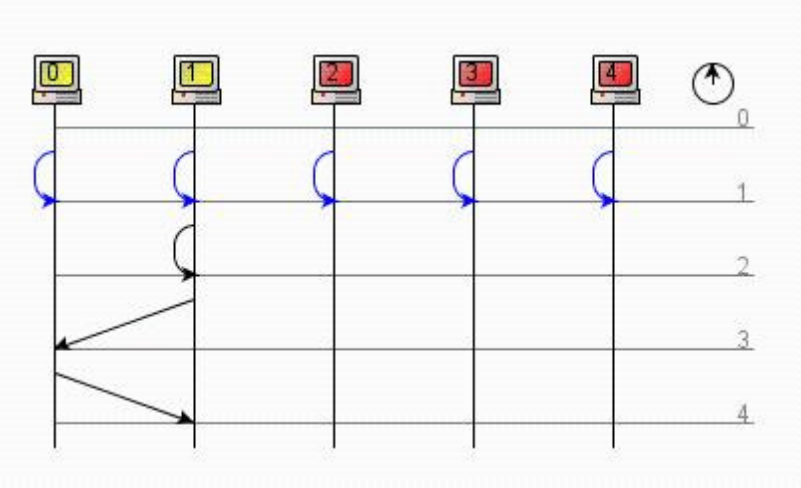

Abb. 6.1. Bezeichnung der Abbildung

6.3 Mathematische Formel

Mathematische Formeln bzw. Formulierungen können sowohl im laufenden Text (z.B. $y = x^2$) oder abgesetzt und zentriert im Text erscheinen. Gleichungen sollten für Referenzierungen nummeriert werden (siehe Formel 6.1).

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$			
P_2		$W(x)2$		
P_3		$R(x)2$		$R(x)1$
P_4			$R(x)2$	$R(x)1$

Tabelle 6.1. Bezeichnung der Tabelle

$$e_i = \sum_{i=1}^{n} w_i x_i \tag{6.1}$$

Entscheidungsformel:

$$\psi(t) = \begin{cases} 1 & 0 <= t < \frac{1}{2} \\ -1 & \frac{1}{2} <= t < 1 \\ 0 & sonst \end{cases} \tag{6.2}$$

Matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \tag{6.3}$$

Vektor:

$$\overline{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \tag{6.4}$$

6.4 Sätze, Lemmas und Definitionen

Sätze, Lemmas, Definitionen, Beweise, Beispiele können in speziell dafür vorgesehenen Umgebungen erstellt werden.

Definition 6.1. *(Optimierungsproblem)*
Ein Optimierungsproblem $\mathcal{P}$ *ist festgelegt durch ein Tupel* $(I_\mathcal{P}, sol_\mathcal{P}, m_\mathcal{P}, goal)$ *wobei gilt*

1. $I_\mathcal{P}$ *ist die Menge der Instanzen,*
2. $sol_\mathcal{P} : I_\mathcal{P} \longmapsto \mathbb{P}(S_\mathcal{P})$ *ist eine Funktion, die jeder Instanz* $x \in I_\mathcal{P}$ *eine Menge zulässiger Lösungen zuweist,*
3. $m_\mathcal{P} : I_\mathcal{P} \times S_\mathcal{P} \longmapsto \mathbb{N}$ *ist eine Funktion, die jedem Paar* $(x, y(x))$ *mit* $x \in I_\mathcal{P}$ *und* $y(x) \in sol_\mathcal{P}(x)$ *eine Zahl* $m_\mathcal{P}(x, y(x)) \in \mathbb{N}$ *zuordnet (= Maß für die Lösung* $y(x)$ *der Instanz* x*), und*
4. $goal \in \{min, max\}$.

Beispiel 6.2. MINIMUM TRAVELING SALESMAN (MIN-TSP)

- $I_{MIN-TSP} =_{def}$ s.o., ebenso $S_{MIN-TSP}$
- $sol_{MIN-TSP}(m, D) =_{def} S_{MIN-TSP} \cap \mathbb{N}^m$
- $m_{MIN-TSP}((m, D), (c_1, \ldots, c_m)) =_{def} \sum_{i=1}^{m-1} D(c_i, c_{i+1}) + D(c_m, c_1)$
- $goal_{MIN-TSP} =_{def} min$

□

Satz 6.3. *Sei $\mathcal{P}$ ein **NP**-hartes Optimierungsproblem. Wenn $\mathcal{P} \in$ **PO**, dann ist **P** = **NP**.*

Beweis. Um zu zeigen, dass **P** = **NP** gilt, genügt es wegen Satz A.30 zu zeigen, dass ein einziges **NP**-vollständiges Problem in **P** liegt. Sei also $\mathcal{P}'$ ein beliebiges **NP**-vollständiges Problem.

Weil $\mathcal{P}$ nach Voraussetzung **NP**-hart ist, gilt insbesondere $\mathcal{P}' \leq_T \mathcal{P}_C$. Sei R der zugehörige Polynomialzeit-Algorithmus dieser Turing-Reduktion. Weiter ist $\mathcal{P} \in$ **PO** vorausgesetzt, etwa vermöge eines Polynomialzeit-Algorithmus A. Aus den beiden Polynomialzeit-Algorithmen R und A erhält man nun leicht einen effizienten Algorithmus für $\mathcal{P}'$: Ersetzt man in R das Orakel durch A, ergibt dies insgesamt eine polynomielle Laufzeit. □

Lemma 6.4. *Aus **PO** = **NPO** folgt **P** = **NP**.*

Beweis. Es genügt zu zeigen, dass unter der angegeben Voraussetzung KNAPSACK ∈ **P** ist.

Nach Voraussetung ist MAXIMUM KNAPSACK ∈ **PO**, d.h. die Berechnung von $m^*(x)$ für jede Instanz x ist in Polynomialzeit möglich. Um KNAPSACK bei Eingabe (x, k) zu entscheiden, müssen wir nur noch $m^*(x) \geq k$ prüfen. Ist das der Fall, geben wir 1, sonst 0 aus. Dies bleibt insgesamt ein Polynomialzeit-Algorithmus.

□

6.5 Fußnoten

In einer Fußnote können ergänzende Informationen[1] angegeben werden. Außerdem kann eine Fußnote auch Links enthalten. Wird in der Arbeit eine Software (zum Beispiel Java-API[2]) eingesetzt, so kann die Quelle, die diese Software zur Verfügung stellt in der Fußnote angegeben werden.

6.6 Literaturverweise

Alle benutzte Literatur wird im Literaturverzeichnis angegeben[3]. Alle angegebene Literatur sollte mindestens einmal im Text referenziert werden[CDK02].

[1] Informationen die für die Arbeit zweitrangig sind, jedoch für den Leser interessant sein könnten.
[2] `http://java.sun.com/`
[3] Dazu wird ein sogennanter bib-File, literatur.bib verwendet.

7

Beispiel-Kapitel

In diesem Kapitel wird beschrieben, warum es unterschiedliche Konsistenzmodelle gibt. Außerdem werden die Unterschiede zwischen strengen Konsistenzmodellen (Linearisierbarkeit, sequentielle Konsistenz) und schwachen Konsistenzmodellen (schwache Konsistenz, Freigabekonsistenz) erläutert. Es wird geklärt, was Strenge und Kosten (billig, teuer) in Zusammenhang mit Konsistenzmodellen bedeuten.

7.1 Warum existieren unterschiedliche Konsistenzmodelle?

Laut [Mal97] sind mit der Replikation von Daten immer zwei gegensätzliche Ziele verbunden: die Erhöhung der Verfügbarkeit und die Sicherung der Konsistenz der Daten. Die Form der Konsistenzsicherung bestimmt dabei, inwiefern das eine Kriterium erfüllt und das andere dementsprechend nicht erfüllt ist (Trade-off zwischen Verfügbarkeit und der Konsistenz der Daten). Stark konsistente Daten sind stabil, das heißt, falls mehrere Kopien der Daten existieren, dürfen keine Abweichungen auftreten. Die Verfügbarkeit der Daten ist hier jedoch stark eingeschränkt. Je schwächer die Konsistenz wird, desto mehr Abweichungen können zwischen verschiedenen Kopien einer Datei auftreten, wobei die Konsistenz nur an bestimmten Synchronisationspunkten gewährleistet wird. Dafür steigt aber die Verfügbarkeit der Daten, weil sie sich leichter replizieren lassen.

Nach [Mos93] kann die Performanzsteigerung der schwächeren Konsistenzmodelle wegen der Optimierung (Pufferung, Code-Scheduling, Pipelines) 10-40 Prozent betragen. Wenn man bedenkt, dass mit der Nutzung der vorhandenen Synchronisierungsmechanismen schwächere Konsistenzmodelle den Anforderungen der strengen Konsistenz genügen, stellt sich der höhere programmiertechnischer Aufwand bei der Implementierung der schwächeren Konsistenzmodelle als ihr einziges Manko dar.

In [Che85] ist beschrieben, wie man sich Formen von DSM vorstellen könnte, für die ein beachtliches Maß an Inkonsistenz akzeptabel wäre. Beispielsweise könnte DSM verwendet werden, um die Auslastung von Computern in einem Netzwerk zu speichern, so dass Clients für die Ausführung ihrer Applikationen die am wenigsten ausgelasteten Computer auswählen können. Weil die Informationen dieser Art innerhalb kürzester Zeit ungenau werden können (und durch die Verwendung der

veralteten Daten keine großen Nachteile entstehen können), wäre es vergebliche Mühe, sie ständig für alle Computer im System konsistent zu halten [CDK02]. Die meisten Applikationen stellen jedoch strengere Konsistenzanforderungen.

7.2 Klassifizierung eines Konsistenzmodells

Die zentrale Frage, die für die Klassifizierung (streng oder schwach) eines Konsistenzmodells von Bedeutung ist [CDK02]: wenn ein Lesezugriff auf eine Speicherposition erfolgt, welche Werte von Schreibzugriffen auf diese Position sollen dann dem Lesevorgang bereitgestellt werden? Die Antwort für das schwächste Konsistenzmodell lautet: von jedem Schreibvorgang, der vor dem Lesen erfolgt ist, oder in der „nahen“ Zukunft, innerhalb des definierten Betrachtungsraums, erfolgten wird. Also irgendein Wert, der vor oder nach dem Lesen geschrieben wurde.

Für das strengste Konsistenzmodell, Linearisierbarkeit (atomic consistency), stehen alle geschriebenen Werte allen Prozessoren sofort zur Verfügung: eine Lese-Operation gibt den aktuellsten Wert zurück, der geschrieben wurde, bevor das Lesen stattfand. Diese Definition ist aber in zweierlei Hinsicht problematisch. Erstens treten weder Schreib- noch Lese-Operationen zu genau einem Zeitpunkt auf, deshalb ist die Bedeutung von „aktuellsten“ nicht immer klar. Zweitens ist es nicht immer möglich, genau festzustellen, ob ein Ereignis vor einem anderen stattgefunden hat, da es Begrenzungen dafür gibt, wie genau Uhren in einem verteilten System synchronisiert werden können.

Nachfolgend werden einige Konsistenzmodelle absteigend nach ihrer Strenge vorgestellt. Zuvor müssen wir allerdings klären, wie die Lese- und Schreibe-Operationen in dieser Ausarbeitung dargestellt werden.

Sei x eine Speicherposition, dann können Instanzen dieser Operationen wie folgt ausgedrückt werden:

- $R(x)a$ - eine Lese-Operation, die den Wert a von der Position x liest.
- $W(x)b$ - eine Schreib-Operation, die den Wert b an der Position x speichert.

7.3 Linearisierbarkeit (atomic consistency)

Die Linearisierbarkeit im Zusammenhang mit DSM kann wie folgt definiert werden:

- Die verzahnte Operationsabfolge findet so statt: wenn $R(x)a$ in der Folge vorkommt, dann ist die letzte Schreib-Operation, die vor ihr in der verzahnten Abfolge auftritt, $W(x)a$, oder es tritt keine Schreib-Operation vor ihr auf und a ist der Anfangswert von x. Das bedeutet, dass eine Variable nur durch eine Schreib-Operation geändert werden kann.
- Die Reihenfolge der Operationen in der Verzahnung ist konsistent zu den Echtzeiten, zu denen die Operationen bei der tatsächlichen Ausführung aufgetreten sind.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$		$W(y)2$	
P_2		$R(x)1$		$R(y)2$

Tabelle 7.1. Linearisierbarkeit ist erfüllt

Die Bedeutung dieser Definition kann an folgendem Beispiel (Tabelle 7.1) nachvollzogen werden. Es sei angenommen, dass alle Werte mit 0 vorinitialisiert sind.

Hier sind beide Bedingungen erfüllt, da die Lese-Operationen den zuletzt geschriebenen Wert zurückliefern. Interessanter ist es, zu sehen, wann die Linearisierbarkeit verletzt ist.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$	$W(x)2$		
P_2			$R(x)0$	$R(x)2$

Tabelle 7.2. Linearisierbarkeit ist verletzt, sequentielle Konsistenz ist erfüllt.

In diesem Beispiel (Tabelle 7.2) ist die Echtzeit-Anforderung verletzt, da der Prozess P_2 immer noch den alten Wert liest, obwohl er von Prozess P_1 bereits geändert wurde. Diese Ausführung wäre aber sequentiell konsistent (siehe kommender Abschnitt), da es eine Verzahnung der Operationen gibt, die diese Werte liefern könnte ($R(x)0, W(x)1, W(x)2, R(y)2$). Würde man beide Lese-Operationen des 2. Prozesses vertauschen, wie in der Tabelle 7.3 dargestellt, so wäre keine sinnvolle Verzahnung mehr möglich.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$	$W(x)2$		
P_2			$R(x)2$	$R(x)0$

Tabelle 7.3. Linearisierbarkeit und sequentielle Konsistenz sind verletzt.

In diesem Beispiel sind beide Bedingungen verletzt. Selbst wenn die Echtzeit, zu der die Operationen stattgefunden haben, ignoriert wird, gibt es keine Verzahnung einzelner Operationen, die der Definition entsprechen würde.

8

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

Literaturverzeichnis

CDK02. Coulouris, George, Jean Dollimore und Tim Kindberg: *Verteilte Systeme: Konzepte und Design*. Addison-Wesley-Verlag, 2002.

Che85. Cheriton, David R.: *Preliminary Thoughts on Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems*, 1985.

Mal97. Malte, Peter: *Replikation in Mobil Computing*. Seminar No 31/1997, Institut für Telematik der Universität Karlsruhe, Karsruhe, 1997. `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=/ira/1997/31`.

Mos93. Mosberger, David: *Memory Consistency Models*. Technical Report 93/11, University of Arizona, November 1993.

A

Glossar

DisASTer	DisASTer (Distributed Algorithms Simulation Terrain), A platform for the Implementation of Distributed Algorithms
DSM	Distributed Shared Memory
AC	Linearisierbarkeit (atomic consistency)
SC	Sequentielle Konsistenz (sequential consistency)
WC	Schwache Konsistenz (weak consistency)
RC	Freigabekonsistenz (release consistency)

B

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

Datum	Unterschrift der Kandidatin / des Kandidaten