UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Konzeption und Implementierung eines Unified Rendering Frameworks mit modernen GPU-Computing-APIs

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Markus Schlüter

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inform. Dominik Grüntjens

Koblenz, im Mai 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .
(Ort, Datum) (Unterschrift)

Todo list

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieser Bachelorarbeit wurde der Frage nachgegangen, inwiefern eine sogenannte „Unified Rendering-Engine", welche verschiedene Simulationsdomänen vereint, einen Mehrwert darstellen kann gegenüber dem klassischen Ansatz, z.B. gesondert sowohl eine Graphik- als auch eine Physik-Engine zu verwenden, die zunächst einmal keinen Bezug zueinander haben;

Hierbei wurde besonderer Wert auf die Verwendung moderner GPU-Computing-APIs gelegt, namentlich auf OpenGL3/4 und OpenCL.
Da bei diesem ganzheitlichen Thema eine vollständige Implementierung einer solchen vereinheitlichten Engine unmöglich war, konnte nur ein Bruchteil der Konzepte implementiert werden;

Dieser Umstand war von Vornherein bekannt, und die Versuchung ist stark, wie in einer Demo die schnelle Realisierung eines Feature-Sets einer konsistenteren, aber zeitaufwändigeren und zunächst karger wirkenden Implementierung vorzuziehen. Dieser Versuchung wurde versucht, nur dort nachzugeben, wo die negativen Auswirkung auf die Konsistenz des Gesamtsystems lokal bleiben, und so nicht „Hacks“ sich irreversibel durch das gesamte System ziehen.

Letztendlich wurden exemplarisch für die Nutzung in der visuellen Simulationsdomäne einige gängige visuelle Effekte einer Graphik-Engine implementiert, wie Shadow Mapping, Normal Mapping, Environment Mapping, Displacement Mapping und dynamisches LOD. Es wurden moderne OpenGL- und Hardware-Features wie Instancing, Uniform Buffers und Hardware-Tesselation verwendet. Schwerpunkt war hier der Einsatz einer Template-Engine, damit

1. Boilerplate-Code in den Shadern vermieden wird und

2. Effekte beliebig (sinnvoll) nach Möglichkeit zur Laufzeit miteinander kombinierbar sind

. Mehr dazu in Kapitel 4.1
In der mechanischen Simulationsdomäne wurde eine partikelbasierte Fluidsimulation mit OpenCL auf Basis von Smoothed Particle Hydrodynamics implementiert. Mehr dazu in Kapitel 4.2.

Das System trägt den Namen „Flewnit“, eine bewusst nicht auf den ersten Blick erkennbar sein sollende[1] Kombination der Worte „Fluid“, in

[1] Es soll der generalistische Ansatz des Frameworks nicht in den Hintergrund gedrängt werden.

Anspielung auf den urspünglichen Zweck einer Bibliothek zur Fluidsimulation und „Unit“, in Anspielung auf „Unity“-„Einheit“. Zufälligerweise ist das Nit auch noch die englische Einheit für die Leuchtdichte, $\frac{Cd}{m^2}$.

1.1 Motivation

Ursprünglich als Arbeit zur Implementierung einer Fluidsimulation geplant, wurde bald ein generalistischer, eher softwaretechnisch orientierter Ansatz verfolgt, der jedoch die Implementierung einer Fluidsimulation als mittelfristiges Ziel hatte;

1.1.1 „Unified Rendering Engine“

Der Wunsch nach einer „Unified Rendering Engineërwächst aus eigener Erfahrung der Kopplung von Physik- und Graphik-Engines, namentlich der Bullet Physics Library[2] und der OGRE Graphik-Engine[3]. Diese Hochzeit zweier Engines, die jeweils für verschiedene „Simulationsdomänen“ zuständig sind, bringt gewissen Overhead mit sich, da Konzepten wie Geometrie und ihrer Transformationen unterschiedliche Repräsentationen bzw. Klassen zugrunde liegen; Hierdurch wird die gemeinsame Nutzung beider Domänen von Daten wie z.B. Geometrie nahezu unmöglich; Ferner müssen für eine die beiden Engines benutzende Anwendung diese Klassen mit ähnlicher Semantik durch neue Adapterklassen gewrappt werden, um dem Programmier der eigentlichen Anwendungslogik den ständigen Umgang mit verschiedenen Repräsentationen und deren Synchronisation zu ersparen.

evtl. beispielschema erstellen für zwei klassischee transformationsklassen und adapter vs. unified transformation

Zitat einfügen? Stefan Müller, PCG? ;)

Die Aussage „Photorealistische Computergraphik ist die Simulation von Licht“ hat mich wohl auch inspiriert, den Simulationsbegriff allgemeiner aufzufassen und das Begriffspaar "Rendering und Physiksimulation“ zu hinterfragen[4].

Es sei bemerkt, dass weder eine Hypothese bestätigt noch widerlegt werden sollte, geschweige denn überhaupt eine (mir bekannte) Hypothese im Vorfeld existierte; Es sprechen etliche Argumente für eine Vereinheitlichung der Konzepte (geringerer Overhead durch Wegfall der Adapterklassen, evtl. Speicherverbrauch durch z.T. gemeinsame Nutzbarkeit von Daten), aber auch einige dagegen (Komplexität eines Systems, Anzahl an theoretischen

[2] http://bulletphysics.org
[3] http://www.ogre3d.org
[4] Auch wenn dieses Framework nicht vornehmlich auf physikalisch basierte, also photorealistische Beleuchtung ausgelegt ist, soll diese aufgrund des generalistischen Konzepts jedoch integrierbar sein.

Kombinationsmöglichkeiten steigt, viele sind unsinnig und müssen implizit oder explizit ausgeschlossen werden).

Für mich persönlich bringt die Bearbeitung dieser Fragestellung zahlreiche Vorteile; Ich muss ein wenig ausholen:
Schon als Kind war ich begeistert von technischen Geräten, auf denen interaktive Computergraphik möglich war; Sie sprechen sowohl das ästhetische Empfinden an, als auch bieten sie eine immer mächtigere Ergänzungs- und Erweiterungsmöglichkeit zu unserer Realität an; Letztendlich stellten diese Geräte für mich wohl auch immer ein Symbol dafür dar, in wie weit die Menschheit inzwischen fähig ist, den Mikrokosmos zu verstehen und zu nutzen, damit demonstriert, dass sie zumindest die rezeptiven und motorischen Beschränkungen seiner Physiologie überwunden hat.
Die Freude an Schönheit und Technologie findet für mich in der Computergraphik und der sie ermöglichenden Hardware eine Verbindungsmöglichkeit; Die informatische Seite mit seinen Algorithmen als auch die technische Seite mit seinen Schaltungen faszinieren mich gleichermaßen; Auch das „große Ganze"der Realisierung solcher Computergraphischen Systeme, das Engine-Design mit seinen softwaretechnischen Aspekten, interessiert mich.
Ferner wollte ich schon immer "die Welt verstehen", sowohl auf physikalisch-naturwissenschaftlich-technischer, als auch - aufgrund der system-immanenten Beschränkungen unseres Universums - auf metaphysischer Ebene[5];

Und hier schließt sich der Kreis: Sowohl in der Philosophie als auch in der Informatik spielt das Konzept der Abstraktion eine wichtige Rolle; Nichts anderes tut eine „Unified Engine": sie abstrahiert bestehende Konzepte teilgebiets-spezifischer Engines, wie z.B. Graphik und Physik; Ich erhoffe mir, dass mit dieser Abstraktion man in seinem konzeptionellen Denken der der realen Welt ein Stück weit näher kommt; Die verfügbaren Rechenressourcen steigen, die Komplexität von Simulationen ebenfalls; Ob eine semantische Generalisierung von seit Jahrzehnten verwendeten Begriffen wie „Rendering“ und „Physiksimulation“, welche dieser Entwicklung angemessen sein soll, eher hilfreich oder verwirrend ist, kann eine weitere Interessante Frage sein, die ich jedoch nicht weiter empirisch untersucht habe.

Letzendlich verbindet dieses Thema also viele meiner Interessen, welche die gesamte Pipeline eines Virtual-Reality-Systems, vom Konzept einer Engine bis hin zu den Transistoren einer Graphikkarte, auf sämtlichen Abstraktionsstufen betreffen: Es gab mir die Möglichkeit,

- den Mehrwert einer Abstraktion gängiger Konzepte von Computergraphik und Physiksimulation zu erforschen

[5] ob der Begriff „Verständnis“ im letzten Falle ganz treffend ist, bleibt Ermessens-Sache

- meine Erfahrung im Engine-Design zu vertiefen
- meine Erfahrungen im (graphischen) Echtzeit-Rendering zu vertiefen
- mich mit Physiksimulation (genauer: Simulation von Mechanik) zu beschäftigen, konkret mit Fluidsimulation
- mich in OpenGL 3 und 4 einzuarbeiten, drastisch entschlackten Versionen der Graphik-API, deren gesäuberte Struktur die Graphikprogrammierung wesentlich generalistischer macht und somit die Abstraktion erleichtert
- mich in OpenCL einzuarbeiten, den ersten offenen Standard für GPGPU[6]
- mich intensiver mit Graphikkarten-Hardware, der zu Zeit komplexesten und leistungsfähigsten Consumer-Hardware zu beschäftigen, aus purem Interesse und um die OpenCL-Implementierung effizienter zu gestalten

Die vielseitigen didaktischen Aspekte hatten bei dieser Themenwahl also ein größeres Gewicht als der Forschungsaspekt, was dem Ziel einer Bachelorarbeit angemessen ist.

1.1.2 Fluidsimulation

Warum ich mich bei der exemplarischen Implementierung einer mechanischen Simulationsdomäne für eine Fluidsimulation entschieden habe, hat viele Gründe:

Muss ich das irgendwie belegen? Ich wüsste nicht, wie/wo ;(

- Wohingegen Rigid Body-Simulation in aktuellen Virtual-Reality-Anwendungen wie Computerspielen schon eine recht große Verbreitung erreicht hat, sucht man eine komplexere physikalisch basierte Fluidsimulation, die über eine 2,5 D-Heightfield-Implementation hinausgehen, noch vergebens; Das Ziel, eine deratige Fluidsimulation in einen Anwendungskontext zu integrieren, der langfristig über den einer Demo hinausgehen soll, hat also etwas leicht pionierhaftes.
- Da Fluide für gewöhnlich ein homogenes Medium sind, eignet sich für die Suche nach Nachbarpartikeln die Beschleunigungsstruktur des Uniform Grid besonders gut, wohingegen sich für Rigid- oder Softbody- Simulation eher komplexere Beschleunigungsstrukturen wie Oct-Trees, Bounding Volume-Hierarchies oder kD-Trees anbieten; Letztere Strukturen lassen sich schwerer auf die GPU mappen, welche

[6] General Purpose Graphics Processing Unit- Computing, die Nutzung der auf massiver Paralleltät beruhenden Rechenleistung von Graphikkarten in nicht explizit Graphikrelevanten Kontexten

als Stream-Prozessor nicht optimal für komplexe Kontrollflüsse und Datenstrukturen geeignet ist. Die Fluidsimulation stellt also sowohl vor dem Hintergrund der GPU-Implementierung als auch auf mathematisch-phyikalischem Gebiet einen relativ „seichten“ Einstieg in die Welt der Echtzeit-Physiksimulation dar, nicht zuletzt, weil es schon zahlreiche Arbeiten zur Fluidsimulation gibt, welche erfolgreich den Spagat zwischen Echtzeitfähigkeit und physikalischer Plausibilität gemeistert haben.

noch eine Referenz? Common knowledge zu belegen, scheint mir so überflüssig, und es fehlt mir auch "DIE"Referenz für solche common-knowledge-Sachen.. Cuda Programming Guide? .... ;(

Warum Fluid? interesse an physik, ßeichter einstieg"durch verinfachte Algorithmen, so lange es "nurüm plausibilität geht und nicht um physikalische korrektheit, einfacheres Mapping der homogh Warum Partikelbasiert? ... theoretisch unendliche simulationsdomäne, direkte darstellungsmöglichkeit als openGL vertices, damit directe nutzung von CL/GL interop und somit gemeinsamer nutzung von daten ind beiden domänen..., einfachere mathematik dank lagrange'scher betrachtung, kein advektionsterm, besser einsetzbar für Liquide, da Voxelbasierte ansätze "tricksen"müssen, um das volumen einer teilmenge eines Fluids zu bbewahren;

2 Überblick

2.1 Vision

Verwendung der Unified Engine zur Entwicklung einer echtzeit-fähigen Paddel-Simulation mit ausgefeilter Fluid-Mechanik und -Visualisierung, rigid bodies, statischem Kollisions-Mesh;

2.2 Begriffe

erklären, was ich unter Unified Rendering verstehe, was Rendering dadruch fuer eine generalistische Bedeutung bekommt;
tabelle mit Gegenüberstellung

2.3 Paradigmen

moeglichst symmetrischer Ansatz zwischen den Simulationsdomaenen, zantralistische Verwaltung von spezifischen Objekten, GPU-Code gernaierung per Template-Engine zur vermeidung von boilerplate, moeglichst versuchen, langfristig so viele Aspekte wie moeglich miteinander kombinieren zu koennern (Visualisierungstechniken, verschiedene Physik Partikelsimulation, Rigid bodiy-Simulation)

potential bewahren, dass letztendlich, in der Entwicklungszeit nach dieser Bachelorarbeit eine möglichst seriöse Lightweight-Unified-Engine entstehen könnte;

2.4 Schwerpunkte

2.4.1 Template-Engine

boilerplate, kombinierbarkeit, nach Möglichkeit lesbarkeit
exemplarischer code schnipsel

2.4.2 Performance duch Implementierung auf der GPU mit modernen GPU-Computing-APIs

auf die massive parallelität eingehen, die sowohl von visualler wie mechanischer domäne genutzt werden kann; Performance-Schwerpunkt, Optimierung, auch hardware-abhängige, erwähnen, Gegenüberstellung zu alten OpenGL-nutzenden GPUPU- Verfahren, die nicht scattern konnten in texturen rendern mussten und auch sonst etliche Nachteile hinnehmen mussten

2.4.3 Potential der Entwicklung hin zu einer Unified Engine nicht verschenken

i.e. fenstermanager, input etc sollten wohl übelegt sein, für vielseitig, flexible anwendung zur Laufzeit sollten keine Speicher-Lecks auftreten, damit Funktionalität kontrolliert heruntergefahren und neu initialisiert werden kann; Für gemeinsamen zugriff sollten viele Daten für andere Klassen verfügbar sein (Buffer, Rendering Results...); Realisierung uber Manager-Singleton-Klassen und Zugriff über Map-Container;

3 Systemarchitektur

3.1 Entwicklungsumgebung

Linux, CMake, git, Eclipse ; gründe für auswahl: linux: paketmanager für dependencies, unzufriedenheit mit microsofts stiefmütterlicher Behandlung von 64bit- programmen, nicht zuletzt wunsch nach Vertiefung der kenntnisse der linux-welt

cross platform build system, moderne versionsverwaltung,

3.2 Klassendiagramm

3.3 Dependencies

3.3.1 OpenGL3/4

3.3.2 OpenCL 1.0

noch keine offenen openCL 1.1 treiber, außerdem keine features davon benötigt; c++-wrapper genutzt, auch von khronos-seiten beziehbar; Gegenüberstellung zu CUDA, vor- und nachteile auflisten, insbesondere das problem ‚dass esk ein 1D-texturen in OpenCL gibt, und man sich entscheiden muss zwischen generischem buffer und Textur, man also nicht hin und her-interpretieren kann wie in CUDA;

3.3.3 GLFW

explizite GL3 core profile creation, einfaches fullscreen, multisampling, mouse grab, alles viel besser als GLUT :)

3.3.4 Grantlee

die string template engine die CL und GL code erzeugt

3.3.5 assimp

3.3.6 ogl math

leichte, aber doch recht maechtige mathe-bibliothek

3.3.7 TinyXML

3.4 Die Buffer-Abstraktion

die bombe, die cpu, ogl und ocl vereint, inclusive ping ponging etc.. fundamentale Klassensammlung fuer den Unified-Aspekt

3.5 Das WorldObject

Basis-Klasse fuer alles was unified simuliert wird: pure viuelle objekt, uniform grid, fluid, rigid body etc..

3.5.1 Das SubObject

3.6 Material

was stellt welches material in welcher Domain dar?

3.7 Geometry

Abtract, Buffer based, Vertex based etc.. ein paar konzepte (implementiert/genutzt nur VertexBased)

3.8 Massively Parallel Program

Basisklasse von Shader und OpenCL Program

3.8.1 Shader

3.8.2 OpenCLProgram

weitere klassen/konzepte to go...

4 Simulation

4.1 Die visuelle Simulationsdomäne

Ein paar worte ueber die shading features, wie sie maskiert werden etc..

4.1.1 Der LightingSimulator

Nochmal drauf hinweisen, dass Rendering etwas generisches in diesem Framework ist, und wir leiber von Lichtsimulation sprechen sollten, auch wenn es monetan nicht photrealistisch ist ;)

4.1.2 Die Lighting Simulation Pipeline Stages

shadowmap gen stage, direct lighting stage, was noch in planung is etc..

4.1.3 ShaderManager

generiert mit grantlee, assigned an materials und verwaltet Shader , abhaenging von der aktuellen lighting stage, den registierten Materials, der Erzeugten kontext, den vom user aktivierten rendering features etc pp

4.1.4 genutzte moderne OpenGL- Features

4.1.4.1 Uniform Buffers

auch von BufferInterface abstrahiert, vorteile auflisten, aber auch stolperfallen: alignment etc)

4.1.4.2 Hardware Tesselation

basics des hardware features erwaehnen fuer den geneigten leser, raptormodell erwaehnen und seinen Aufbereitungsprozess, LOD, displacement mapping erlaeutern

4.1.4.3 Instancing

InstanManager, InstangedGeometry vorstellen, konzept, wie es verwaltet wird, erklaeren

continue brainstorming

4.2 Die mechanische Simulationsdomäne

blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext

4.2.1 Fluidsimulation

blindtext blindtext

überlegen, ob ich aus Interesse nicht noch weiter in die Richtung recherchieren sollte, da ich nach meiner Implementierung erst so richtig beeindruckt von dem Verfahren war (ich habe im Internet noch keine Fluid-Demo gefunden, die ebenfalls SPH implementiert; ok., ich hab auch nicht gesucht ;)), und gerne mehr über die Hintergründe verstehen würde... problem, wie immer: Zeitdruck ;(

4.2.1.1 Grundlagen

blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext

4.2.1.1 .1 Die Navier-Stokes-Gleichungen

Herleitung, Erläuterung blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext

4.2.1.1 .2 Grid-basierte vs. Partikelbasierte Simulation

blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext blindtext

4.2.1.1 .2.1 Die zwei Sichtweisen: Lagrange vs. Euler

blindtext blindtext blindtext blindtext blindtext blindtext

4.2.1.1 .3 Smoothed Particle Hydrodynamics

ursprünglich aus astronomie blubb blubb

4.2.1.2 Verwandte Arbeiten

Referenzen auf Müller03, Thomas Steil, Goswami, GPU gems, Aufzeigen, was ich von wem übernommen habe, was ich selbst modifiziert habe aufgrund von etwaigen Fehlern in den Papers odel weil OpenCL es schlciht nicht zulässt;

4.2.1.3 Umfang

Abgrenzungf zwischen bisher funktionierenden Features, bisher programmierten, aber nicht integrierten und ungetesten Features und TODOs für die zukunft

4.2.1.4 Algorithmen

Verwaltung der Beschleunigungsstruktur ist der Löwenanteil, nicht die physiksimulation, die eher ein Dreizeiler ist;

4.2.1.4 .1 Work Efficient Parallel Prefix Sum

4.2.1.4 .2 Parallel Radix Sort und Stream Compaction

4.2.1.4 .3 Ablauf

initialisierung, und beschreibung der einzelnen phasen...

5 Ergebnisse

6 Ausblick

7 Fazit