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 konsistenten, 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.

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

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

[2] http://bulletphysics.org

[3] http://www.ogre3d.org

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

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.

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 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

[4] Auch wenn dieses Framework nicht vornehmlich auf physikalisch basierte, also photorealistische Beleuchtung ausgelegt ist, soll diese aufgrund des generalistischen Konzepts jedoch integrierbar sein.

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

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
- 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

[6]General Purpose GPU Computing, die Nutzung der auf massiver Paralleltät beruhenden Rechenleistung von Graphikkarten in nicht explizit Graphik-relevanten Kontexten

2 Begriffe

erklaren, was ich unter Unified Rendering verstehe, was Rendering dadruch fuer eine generalistische Bedeutung bekommt;

3 Systemarchitektur

3.1 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)

3.2 KlassenDiagramm

3.3 Dependencies

3.3.1 OpenGL3/4

3.3.2 OpenCL 1.0

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

3.4 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.. erwaehnung des SubObjects;

3.6 Materials

was stellt welches material in welcher Domain dar?

3.7 Geometry

Abtract, Buffer bused, Vertex based etc.. ein paar konzepte (implementier/genutzt nut 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 Domäne

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

4.2 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 ;)

4.3 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

4.4 Instancing

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

4.5 Uniform Buffers

auch von BufferInterface abstrahiert, vorteile auflisten, aber auch stolperfallen 8alignment etc)

4.6 Tesselation

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

4.7 Die mechanische Domäne

Bla bla

4.8 Die Meachanische Domäne

blubb

5 Ergebnisse

Bla bla

5.1 Motivation

blubb

6 Ausblick

Bla bla

6.1 Motivation

blubb

7 Fazit

fazit blubb