Fachhochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

Entwicklung eines JOGL-basierten Frameworks zur Darstellung von Game Physics

- Kurzfassung (Paper) der Diplomarbeit -

ausgearbeitet von:

Adrian Dietzel
Matrikelnummer: 11031862 1 5

im Studiengang Allgemeine Informatik

Ziele und Inhalt der Diplomarbeit

Im Umfang der Diplomarbeit wird ein Rahmenwerk (Framework) für das Wahlpflichtfach „Spiele, Simulation, dynamische Systeme“ erstellt, in dem mittels Java und OpenGL mit *wenig Aufwand* komplexe Spiele-Physiksimulationen (Game Physics) erstellt werden können. Das Framework basiert hierbei auf dem Java-Binding JOGL, welches eine Schnittstelle zwischen Java und OpenGL darstellt. Zwar existiert bereits ein Framework, allerdings lässt dieses nur die Programmiersprache C/C++ zu. In der Fachhochschule Köln wird aber mittlerweile der Fokus auf Java gelegt, wodurch die Anwender (Studenten) des bisherigen Frameworks oftmals Einarbeitungsschwierigkeiten in C/C++ haben. Da das neue Framework in Java entwickelt wird, sind ausführliche Performanceuntersuchungen bezüglich des alten und neuen Frameworks ein weiterer Bestandteil der Diplomarbeit. Auch wird der Aspekt der Portierungsmöglichkeit von bereits vorhandenen C/C++ -Physiksimulationen in das neue Framework genauer unter die Lupe genommen. Abschließend wird exemplarisch eine dynamische 2D-Rauch Simulation basierend auf „Real-Time Fluid Dynamics“ im neuen Framework realisiert, welche sehr gut in Computerspielen eingesetzt werden kann.

Schlüsselworte

Framework, OpenGL, Grafikprogrammierung, JOGL, Java-Binding, Java Native Interface, Game Physics, Performanceuntersuchung, Portierung, 2D-Rauch, Real-Time Fluid Dynamics

1 Einleitung

Der rasante Fortschritt in der Entwicklung von Computersoftware und Computerhardware macht es möglich, dass heutzutage komplexe Multimedia-Anwendungen auf fast jedem aktuellen Heimcomputer einsetzbar sind. Besonders im Bereich der Computerspiele gibt es Fortschritte wie nie zuvor. So ist fast jedes neue Computerspiel mit realistisch aussehender 3D-Grafik ausgestattet, welche zudem noch meist mit umfangreichen Physikberechnungen verknüpft ist. Erst diese Kombination von Spiele-Physik und 3D-Computergrafik lässt Spielewelten richtig lebendig wirken und täuscht eine reale Welt vor. Mit dem zu entwickelnden Framework soll es nun möglich sein, Simulationen solcher Game Physics sehr *einfach* und *komfortabel* in Java zu erstellen. Ein altes C/C++ Framework soll hierbei ersetzt werden, um den Fokus ganz auf die Programmiersprache Java legen zu können. Für die Realisierung des Frameworks wird dabei die Grafikbibliothek OpenGL und das Java-Binding JOGL verwendet werden. Was genau OpenGL und JOGL sind, wird in Abschnitt 2 und 3 kurz erläutert. Darauf wird in Abschnitt 4 beschrieben, wie das Framework aufgebaut ist und welche wichtigen Funktionalitäten es bietet. Die weiteren Abschnitte beinhalten dann Ergebnisse zu Performanceuntersuchungen, wichtige Portierungsaspekte und eine exemplarische Vorstellung einer konkreten Physik-Simulation mit „Real-Time Fluid Dynamics“.

2 Kurzbeschreibung von OpenGL

OpenGL ist die Abkürzung für Open Graphics Library und ist eine *Spezifikation* für ein plattform- und programmiersprachenunabhängiges API zur Entwicklung von 3D-Computergrafik. Die eigentliche Implementierung von OpenGL wird dabei in C realisiert und ist meist in der Treibersoftware der Grafikkarte enthalten. Neben OpenGL gibt es noch eine weitere umfangreiche Bilbliothek für die Grafikdarstellung, welche sich DirectX nennt. Sie wird von Microsoft entwickelt, ist aber nicht plattformunabhängig und somit auf das Betriebssystem Windows eingeschränkt. Aus diesem Gund stellt OpenGL die erste Wahl für die Grafikausgabe des neuen Frameworks dar. [1]

3 Kurzbeschreibung von JOGL

JOGL (Java OpenGL) ist ein sogenanntes Java Binding und erlaubt es, OpenGL mit der Programmiersprache Java zu verbinden. Dabei stellt es eine externe Java-Bibliothek dar, über die ein Zugriff auf die OpenGL-Funktionen, welche in nativem Code vorliegen, möglich ist. Der Zugriff geschieht dabei über das JNI (Java Native Interface), welches die Schnittstelle zwischen nativen Code und plattformunabhängigen Java-Code darstellt.[2] Neben JOGL gibt es noch zahlreiche weitere Java Bindings, wie beispielsweise LWJGL, Mesa3D und GL4Java. Da aber GL4Java und Mesa3D nicht mehr dem aktuellen Stand entsprechen, sind diese bewusst aussortiert worden. LWJGL wäre eine gute Alternative zu JOGL gewesen, lässt sich aber nicht in Verbindung mit einer grafischen Java-Oberfläche wie Swing/AWT verwenden. Auch Java3D wäre als Alternative denkbar, doch da das Framework möglichst einfach in der Handhabung sein soll, käme die komplexe Java3D-API diesem nicht zu gute.

4 Aufbau und Funktionalitäten des Frameworks

Das Framework selber besteht aus 26 Klassen, welche kompakt in einen JAR-Archiv verstaut sind.

[1] Weiterführende Literatur: [ND97], [Ron98] und [Com07]
[2] Umfangreiche JOGL-Informationen: [Dev07b] und [Dev07a]

Dabei besitzt es 5 Komponenten, welche für das Erstellen einer Physiksimulation oft benötigt werden:

1. **Anzeigemenü**
   Es bietet eine individuelle Auswahlmöglichkeit von Auflösung, Farbtiefe und Frequenz des Bildschirms. Auch ein Wechseln zwischen Vollbildmodus (Fullscreen Exclusiv Mode[3]) und Fenstermodus ist möglich. Zudem gibt das Anzeigemenü nützliche Informationen über die Grafikkonfiguration des Systems aus, und auch wird überprüft, ob JOGL korrekt installiert wurde.

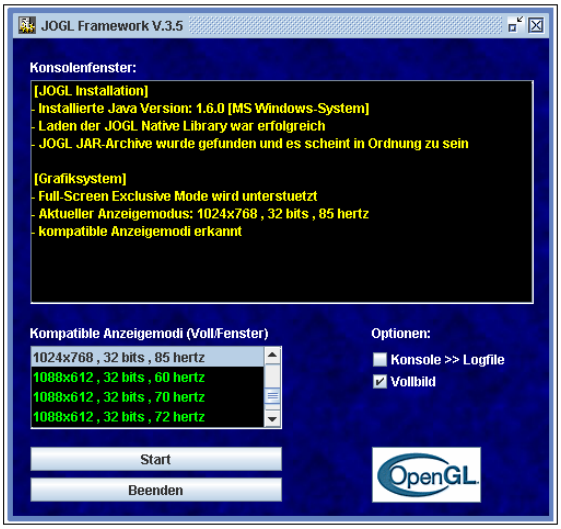

Abbildung 1: Das Anzeigemenü

2. **Maus-/Keyboardhandler**
   Viele Physik-Simulationen erfordern eine Interaktion durch den Benutzer. Dies ist mit dem Maus- und Keyboardhandler des neuen Frameworks leicht möglich. Zudem besitzt der Maushandler die Fähigkeit, relative Mauskoordinaten auszuwerten. Dies ist zum Beispiel wichtig, wenn freie Mausbewegungen, welche nicht auf den Bildschirm eingeschränkt sein dürfen, benötigt werden (Bsp: Umschauen im 3D-Raum mit der Maus).

[3]Siehe: [Mic07]

3. **Dynamische 3D-Kamera**
   Hiermit kann sich der Betrachter in einer Physiksimulation frei umherbewegen und die Simulation aus beliebigen Ansichten anschauen. Sie funktioniert dabei genauso wie die Kamera in dem bekannten Computerspiel „Counter Strike“.

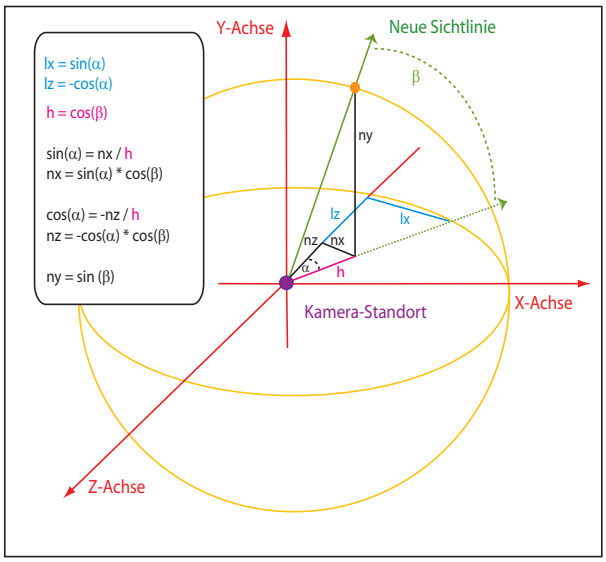

Abbildung 2: Bsp.: Kamera-Rundumblick

4. **FpsCounter**
   Oft wird in Physiksimulationen mit komplexen Algorithmen gearbeitet. Um eine Simulation im Bezug auf die Performance zu optimieren ist eine Zeitmessungsroutine nötig. Diese wird in Form eines FpsCounters bereitgestellt, der Schleifendurchgänge pro Sekunde (fps) im Nanosekundenbereich messen kann.

5. **Textausgabe im Grafikmodus**
   Eine Möglichkeit zum Ausgeben von Text *in der Simulation* ist wichtig, um diese schnell debuggen zu können.

Die einzelnen Komponenten können dabei optional und unabhängig voneinander verwendet werden. Um schnell und einfach eine Physiksimulation programmieren zu können, stellt das Framework zudem bereits 2 fertige, vordefinierte Klas-

sen zur Verfügung, in denen die Physiksimulation programmiert werden kann. Es handelt sich dabei einmal um den Lightweight- und den Heavyweight-Listener.

1. **HeavyweightListener**
   Diese Klasse stellt ein fertiges Grundgerüst dar, in dem alle OpenGL/JOGL-Spezifischen Initialisierungen vorgenommen werden und bereits *alle* Komponenten des Frameworks eingebettet und voreingestellt sind. Die Klasse kann direkt kompiliert und ausgeführt werden. Sie ist für Anfänger gedacht, die weniger mit dem Framework vertraut sind. Zudem besitzt diese einen grafischen 3D-Ursprung, welcher als Orientierungshilfe im 3D-Raum dient. Auch ein Informationsmenü bezüglich der Steuerungshandhabung wird automatisch angezeigt.

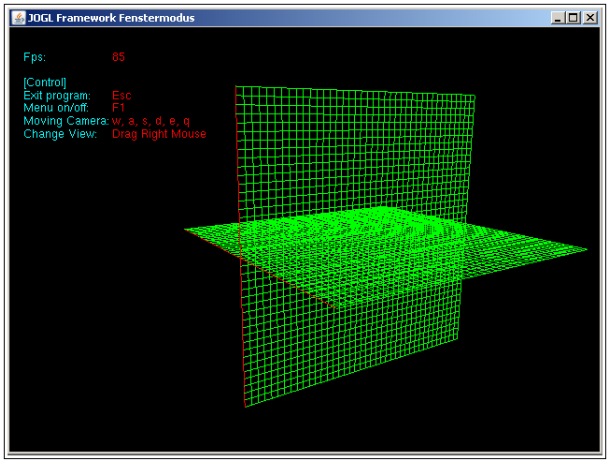

Abbildung 3: HeavyweightListener

2. **LightweightListener**
   Der LightweightListener ist für versierte Benutzer gedacht, die das Framework bereits kennen. Er gleicht in allen Punkten dem HeavyweightListener, nur das dieser alleinig die Keyboardhandler-Komponente implementiert. Somit wird der Benutzer nicht durch Programmcode von Komponenten behindert, welchen er garnicht benötigt.

5 Portierungen von C/C++ Physiksimulationen

Um zu testen, ob das neue Framework für den Einsatz im echten Betrieb geeignet ist und C/C++ Physiksimulationen des alten Frameworks leicht in das neue Framework zu portieren sind, wurde eine solche Portierung mit einer vorhandenen Planeten-Simulation[4] durchgeführt. Diese besitzt umfangreiche physikalische Berechnungen, Maus- und Keyboardsteuerung sowie eine dynamische Kameraroutine. Es stellte sich heraus, dass eini-

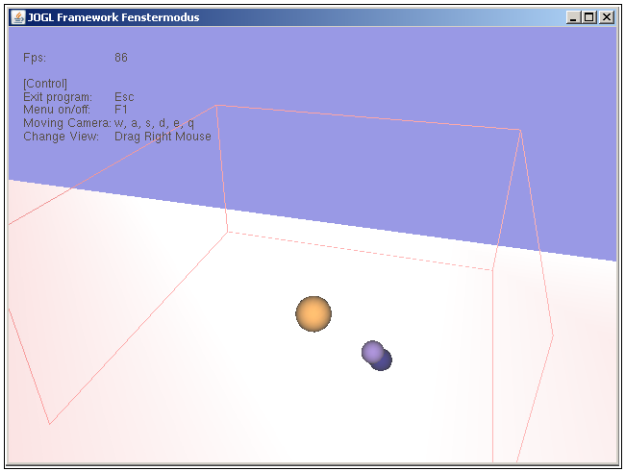

Abbildung 4: Planeten-Simulation im neuen Framework

ge wichtige Punkte bei Portierungen von C/C++ Programmen nach Java zu beachten sind:

- **Makros**
  Da C/C++ im Gegensatz zu Java eine Verwendung von Makros ermöglicht, müssen diese in Java aufgelöst werden.

- **Pointer**
  Da Java keinen direkten Zugriff auf den Speicher erlaubt, können dort keine Pointer benutzt werden. Sind in dem zu portierenden Programm Pointer vorhanden, muss deren

[4] Mond-Erde System, bei dem ein Mond mit 2-3 Schwerpunkten und einer bestimmten Startgeschwindigkeit um die Erde kreist

Funktion herausgefunden werden und entsprechende Äquivalente in Java benutzt werden.

- **Structs**
  Werden structs in C/C++ verwendet, müssen diese in Java als Klasse umgesetzt werden. Bei Zuweisungen von struct zu struct ist zu berücksichtigen, dass C/C++ hier mit „tiefen Kopien“ arbeitet.

Auch ist generell bei der Verwendung von JOGL zu beachten, dass einige OpenGL-Funktionen anders parametrisiert werden als in C/C++. Dies liegt daran, dass einige OpenGL-Funktionen Pointer als Parameter erwarten. Hierzu ein Beispiel jeweils in C/C++ und Java:

```
// *** In C/C++ ***
GLfloat LTest[]={0.5f,0.5f,0.5f,1.0f};
glLightfv(GL_LIGHT0,GL_DIFFUSE,LTest);

// *** In Java/JOGL ***
float LTest[]={0.9f,0.7f,0.7f,1.0f};
gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE,
                  FloatBuffer.wrap(LTest));
//oder
float LTest[]={0.9f,0.7f,0.7f,1.0f};
gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE,
                                LTest, 0);
```

6 Performancetests

Im Rahmen der Performanceuntersuchungen wurden abgestimmte Tests gemacht, welche die Geschwindigkeit des alten und des neuen Frameworks miteinander vergleichen. Da diese sehr spezifisch auf bestimmte Schwerpunkte zugeschnitten sind, wurde zusätzlich ein Performancetest mit einer richtigen Physiksimulation durchgeführt, um das Geschwindigkeitsverhalten in der Praxis zu untersuchen. Auch wurde auf verschiedenen Rechnern im Mathelabor der Fachhochschule Köln getestet, welche Rechnervorraussetzungen nötig sind, um ein moderates Arbeiten mit dem Framework zu ermöglichen.

1. **Vergleich des altes und neues Frameworks**

   - **Aufrufgeschwindigkeit des JNI**
     Hier stellte sich heraus, dass bei einer gezielten Belastung des JNI (66K Kurzlebige OpenGL-Befehle) Java um ca. 63% langsamer ist, als das C/C++ Programm.

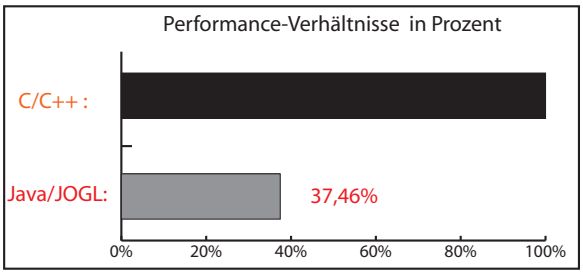

Abbildung 5: Performance-Verhältnis (JNI)

   - **OpenGL-Performance**
     Hier wurde getestet, wie schnell die Ausführungsgeschwindigkeit von OpenGL ist. Diese sollte bei beiden Frameworks gleich schnell sein, wenn die OpenGL-Befehle die Grafikkarte komplett auslasten. Das Testergebnis bestätigt dies:

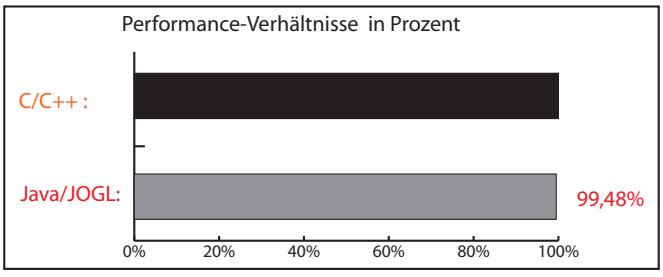

Abbildung 6: Performance-Verhältnis (OpenGL)

   - **Java vs. C/C++**
     Wichtig war auch zu untersuchen, wie schnell C/C++ im Gegensatz zu Java

ist. Da Java einen Hotspot-Compiler besitzt, wurde angenommen, dass die Geschwindigkeit beider Programmiersprachen sich wenig unterscheidet. Es stellte sich heraus, dass Java bei Programmen mit einfachen Schleifen, in denen beispielsweise eine float-Variable manipuliert wird, sogar um 5% schneller ist, als ein C/C++ Programm. Bei Arrayzugriffen aber büßt Java ein Drittel an Performance ein. Dies liegt sehr wahrscheinlich daran, das Java im Gegensatz zu C/C++ noch Überprüfungen bezüglich der Arraygrenzen durchführt.

2. **Vergleich anhand einer Simulation aus der Praxis**
   Da die vorangegangenen Tests ein Extrem darstellen und selten in der Praxis vorkommen, wurde die Performance einer richtigen Physiksimulation analysiert. Dabei wurde die Planetensimulation mit dem Mond-Erde System verwendet und ein wenig erweitert, so dass dort nun 10 unabhängige Monde um die Erde rotieren. Da für das Zeichnen der Monde der Befehl `glutSolidSphere(...)` aus der GLUT-Bibliothek verwendet wurde und dieser bei JOGL überdurchschnittlich langsam ist, schafft das JOGL-Programm nur 20% der Performance des C/C++ -Programmes. Bei weglassen dieses Befehls aber steigt die Performance auf 53%, welches aufgrund des Verhältnisses von JNI-Aufrufen und Arrayzugriffen einen realistischen Wert darstellt.

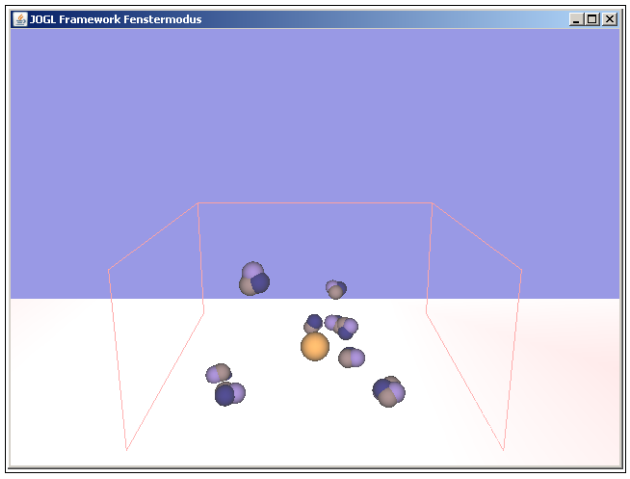

Abbildung 7: 10 Monde (JOGL Framework)

3. **Rechnervorraussetzungen des neuen Frameworks**
   Zuletzt wurde untersucht, auf welchen Rechnern des Mathelabors das Framework generell eingesetzt werden kann und welche Rechner ungeeignet sind. Neben den fps bei einfachen Animationen (rotierendes Dreieck) wurde zudem auch die Antwortzeit der GUI des dazugehörigen Java Code-Editors (JCreator) gemessen. Es stellte sich heraus, dass nur 3 Rechner zu langsam für ein moderates Arbeiten mit dem Framework sind. Alle anderen Systeme erfüllten die Performanceanforderungen. Somit sollte einem Einsatz des neuen JOGL Frameworks nichts mehr im Wege stehen.

7 Real-Time Fluid Dynamics (2D-Rauch)

Abschließend wurde eine interessante 2D-Rauch Simulation mithilfe des neuen Frameworks realisiert. Als Grundlage hierfür diente ein Artikel[5] von Jos Stam zum Thema „Real-Time Fluid Dynamics“. Er zeigt, wie flüssige Strombewegungen (fluid flows) programmiert werden können, welche sogar in Echtzeit (Real-Time) dynamisch beeinflussbar sind. Dabei bedient sich der bereits in C vorliegende Algorithmus den Navier Stoke Glei-

[5] Siehe [Sta07]

chungen:

$$\frac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + v\nabla^2\boldsymbol{u} + \boldsymbol{f} \quad (1)$$

$$\frac{\partial p}{\partial t} = -(\boldsymbol{u} \cdot \nabla)p + k\nabla^2 p + S \quad (2)$$

Mit diesen Gleichungen lässt sich ein dynamischer 2D-Rauch erzeugen, welcher intern mit einem Geschwindigkeitsvektorfeld realisiert wird. Dabei ist jedem Pixel des Bildschirms ein Vektor dieses Feldes zugeordnet. Ein Pixel besitzt dabei Farbwerte zwischen 0 und 1, welche eine Masse/Dichte darstellen. Mit Hilfe der Navier-Stoke Gleichungen kann nun ein über einen bestimmten Zeitraum (t) stattfindender Konzentrationsaustausch der Dichten (ρ) berechnet werden. Diese Diffusion, abhängig vom Geschwindigkeits-Vektorfeld, stellt schließlich eine Illusion von strömenden 2D-Rauch dar.

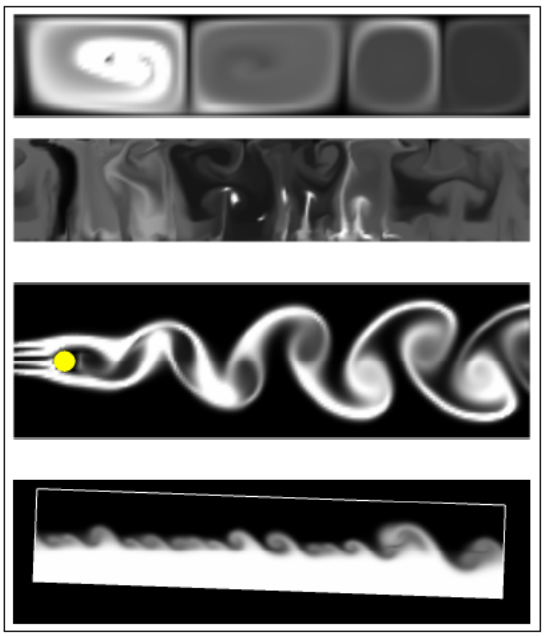

Abbildung 8: Beispiele: 2D-Rauch

8 Fazit

Das Ziel, ein JOGL-basiertes Framework zu entwickeln, welches dem späteren Benutzer die Einarbeitungszeit in OpenGL mit JOGL erspart, konnte nach einigen Anlaufschwierigkeiten komplett erreicht werden. Zwar hat sich herausgestellt, dass das neue JOGL-basierte Framework in etwa nur halb so performant ist, wie das bisherige Framework, für einen praktischen Einsatz ist es aber vollkommen ausreichend. Außerdem besitzt das neue Framework eine übersichtliche, objektorientierte Programmstruktur, und die Möglichkeit der Verwendung von umfangreichen neuen Funktionen, wie beispielsweise einer dynamischen 3D-Kamerasteuerung, lassen die Vorteile bei weitem überwiegen. Die Diplomarbeit hat gezeigt, dass Java längst nicht mehr nur für Server-Programme oder einfache GUI-Anwendungen geeignet ist. Durch die OpenGL-Bindings ist Java nun in der Lage, sich auch in der Grafik-/Spieleprogrammierung behaupten zu können. Auch wird festgestellt, dass „Game Physics“, welche auf dem neuen Framework simuliert werden sollen, mittlerweile so wichtig und unentbehrlich sind, dass einige neue PC-Systeme (Bsp.: Firma Alienware) sogar standardmäßig mit einem zusätzlichen Physik-Chip[6] ausgestattet werden.

[6] AGEIA PhysX Chip = erster Physik-Prozessor der Welt <http://www.ageia.com>

Literatur

[Com07] Community: *Neon Helium Productions*. Version: 2007. <http://nehe.gamedev.net/>, Abruf: 14.03.2007

[Dev07a] Developer, JOGL: *JOGL API Dokumentation*. Version: 2007. <http://download.java.net/media/jogl/builds/archive/jsr-231-1.1.0-rc2/jogl-1.1.0-rc2-docs.zip>, Abruf: 14.03.2007

[Dev07b] Developer, JOGL: *JOGL Official Users Guide*. Version: 2007. <https://jogl.dev.java.net/unbranded-source/browse/*checkout*/jogl/doc/userguide/index.html>, Abruf: 14.03.2007

[Mic07] Microsystems, Sun: *Java Tutorials - Full-Screen Exclusive Mode*. Version: 2007. <http://java.sun.com/docs/books/tutorial/extra/fullscreen/exclusivemode.html>, Abruf: 14.03.2007

[ND97] Neider, Jackie ; Davis, Tom: *OpenGL Redbook 1.1*. Version: 1997. <http://www.gamedev.net/download/redbook.pdf>, Abruf: 14.03.2007

[Ron98] Ron, Fosner: *OpenGL Programming for Windows95 and Windows NT*. 6. Auflage. Addison Wesley, 1998

[Sta07] Stam, Jos: *Real-Time Fluid Dynamics for Games*. Version: 2007. <http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/GDC03.pdf>, Abruf: 14.03.2007