BunnySuite - Ein Framework für 2D-Grafik-Performance-Tests

Ausarbeitung zum Serious Games Praktikum

Maximilian Li, Victor Ferdinand Schümmer, Patrick Pauli

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik
Fachbereich Informatik (Zweitmitglied)

Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Ausarbeitung zum Serious Games Praktikum ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die schriftliche Fassung stimmt mit der elektronischen Fassung überein.

Darmstadt, den 13. September 2016 Maximilian Li, Victor Ferdinand Schümmer, Patrick Pauli

Contents

1 Einleitung **2**

2 Zeitplan und Aufgabenverteilung **3**

3 Grundlagen **4**
3.1 2D-Frameworks 4
3.2 Benchmarking 5

4 Konzept und Design **6**
4.1 Allgemeines 6
4.2 Identifizierte Bottlenecks 6

5 Umsetzung und Tests **7**
5.1 Ablauf der Tests 7
5.2 Beschreibung der Tests 8

6 Zusammenfassung und Ausblick **9**

Bibliography **9**

Abstract

TODO

1 Einleitung

Moderne Computerspiele setzen basieren meistens auf der graphischen Darstellung ihrer Spielwelt. Dabei geht vor allem in den letzten Jahren der Trend verstaerkt in die Richtung photorealistischen 3D-Renderings und weg von der simpleren 2D Darstellung. Dabei ist schnelles 2D-Rendering immernoch eine nichttriviale Aufgabe für GPUs. Für viele Anwendungen sind die verfügbaren Bibliotheken im Allgemeinen immer noch viel zu langsam. Um das zu verbessern, haben wir mit *BunnySuite* ein Framework entwickelt, mit dem gängige 2D-Grafik-Bibliotheken einem automatisierten Stresstest ausgesetzt werden können.
Mit dem *BunnyMark*[1] gab es bereits eine Metrik zum Vergleich von Bibliotheken. Jedoch wurde hier nur die Anzahl der gerenderten Objekte (in diesem Fall Häschen, deshalb der Name) gemessen, bei denen noch 60 fps erreicht werden. Diese Zahl war nur schwer zu interpretieren und musste für ein ausführlicheres Ranking durch aussagekräftigere Metriken, bspw. *Renderzeit pro Frame bei X Objekten* ersetzt werden. Auch findet diese Messung bei BunnyMark zu Demozwecken nur interaktiv statt: Häschen werden per Mausklick zur Szene hinzugefügt. Im von uns entwickelten BunnySuite-Framework können solche Messungen automatisiert erfolgen. Die Tests für mehrere Frameworks werden automatisch nacheinander gestartet und das Ergebnis wird in einem Diagramm zusammengefasst. Das Framework ist leicht erweiterbar, so dass man es mit wenig Aufwand an neue Bibliotheken anpassen kann.
Das vollautomatisierte Test-Framework soll Entscheidern der Spieleentwicklung die Möglichkeit eines Rankings bieten, um die richtige Bibliothek für ihre Anforderungen zu finden. Des Weiteren sollen Entwickler von Bibliotheken das Framework nutzen können, um ihre eigene Engine zu testen und zu optimieren. Die Leistungen der Bibliotheken können so transparent verglichen werden, was den Wettbewerb zwischen den Bibliotheken stimulieren und Anreize setzen soll, stärker an der Performanz zu arbeiten.

[1] https://github.com/openfl/openfl-samples/tree/master/demos/BunnyMark

2 Zeitplan und Aufgabenverteilung

Das Team hat sich darauf geeinigt, dass jeder Entwickler sich auf eine Bibliothek konzentriert. In regelmäßigen Meetings werden Konzepte diskutiert und Designentscheidungen getroffen. Auf diese Weise soll in allem Frameworks eine vergleichbare Implementierung erreicht werden.

	1 PT = 8h		Wer?	
Arbeitspakete	Summe (Personentage)	Viktor	Max	Patrick
1 Management und Koordination	6	2	2	2
1.1 Controlling	3	1	1	1
1.2 Koordination	3	1	1	1
2 Recherche	22	7	7	8
2.1 Grafik-Libraries einarbeiten	4	1	1	2
2.2 Einarbeitung in Bibliotheken	3	1	1	1
2.3 Recherche nach Bottlenecks	15	5	5	5
3 Konzept	18	6	6	6
3.1 Design des Frameworks	4	1	1	2
3.2 Design der Tests	14	5	5	4
4 Implementierung	20	7	7	6
4.1 Implementierung des Frameworks	4	1	1	2
4.2 Implementierung der Tests	16	6	6	4
4.2.1 Tests für LibGDX	4	4	0	0
4.2.2 Tests für SDL	4	0	0	4
4.2.3 Tests für kha	4	2	2	0
4.2.4 Tests für XNA Monogame	4	0	0	0
5 UI	2	1	1	0
6 Ausarbeitung	8	2	2	3
Gesamtaufwand	75	25	25	25

3 Grundlagen

3.1 2D-Frameworks

Game-Frameworks sollen die Entwickler beim Implementieren der Game-Loop unterstützen. Diese Game-Loop besteht hauptsächlich aus 2 Phasen:

Update() Der interne Zustand der Spielwelt und ihrer Objekte wird aktualisiert, bspw. die Position der Spielfigur gemaess der Benutzereingaben angepasst oder das nächste Frame in einer Animation wird ausgewählt oder die Punktzahl erhöht. Zeitgleich finden hier auch komplexere Berechnungen statt, wie die Kollisionsbestimmung, die wieder zu neuen Berechnungen fuehren kann (Positionen werden angepasst, damit Objekte sich nicht ueberlagern; "Gesundheitswerte" werden bei Treffern verringert)

Draw() Die im vorherigen Schritt berechnete Spielwelt wird auf dem Bildschirm dargestellt. Dazu gehören die korrekte Position aller Objekte, die Texturierung und Beleuchtung dieser Objekte, und ggf. Transformationen (Skalierung, Rotation). Auch muss bei Überlagerung von Objekten auf die korrekte Darstellung geachtet werden. Anschließend wird noch die Benutzeroberfläche über den Bildschirminhalt gelegt.

Die Game-Frameworks unterstützen die Entwickler, indem sie verschiedene Funktionalitäten bereits zur Verfügung stellen. Dazu gehören:

- Einfacher Zugriff auf Eingabegeräte, wie Maus, Tastatur, Gamepad, Joystick, Touch- und Gestensteuerung
- optimierte Klassen für Assets wie Audiodateien oder Texturen inklusive Kompression
- optimierte Renderfunktionen, die direkt Grafikschnittstellen wie OpenGL, Direct3D oder Vulkan ansprechen. Diese können auch bereits Transformationen wie Drehungen oder Einfärbungen anbieten.

Diese Bibliotheken helfen den Entwicklern, eine effiziente Implementierung zu erreichen, ohne sehr hardwarenahe Optimierungen selber programmieren zu müssen. Somit soll sehr einfach der "Goldstandard" von 60 Frames per Second in der Darstellungen erreicht werden, bei dem für jeden Schritt der Gameloop eine Rechenzeit von knapp 16,67 ms zur Verfügung steht. Dies beinhaltet das Einlesen von Assets wie Texturen vom Speichersystem der jeweiligen Plattform.
Der wahrscheinlich größte Unterschied zu 3D-Frameworks besteht in der Komplexität der Grafikberechnung. So muss das 2D-Framework keine Informationen über Oberflächen der Objekte, z.B. Orientierung über NormalMaps oder Materialeigenschaften für das *Physically Based Rendering (PBR)*, außer der konkreten Farbe kennen.
Auch können diese Objekte bereits ihrer Darstellung entsprechend als Sprite vorliegen, die, ggf. noch mit Transformationen angepasst, direkt dargestellt werden können. Somit muss kein komplexes dreidimensionales Objekt auf einmal im Speicher gehalten werden, bei dem man sowieso nur die der Kamera zugewandten Seite sieht, sondern nur ein Spritesheet mit den 4 möglichen Orientierungen des Objekts. Ebenso entfällt bei der Darstellung eine komplexe Berechnung der Überlagerung oder verdeckter Rückseiten der Objekte. Es wird nur die

momentane Orientierung der Objekte dargestellt. Die Verdeckung von Objekten kann durch layerbasiertes Rendering gelöst werden:

1. Hintergrund rendern
2. Nicht Spieler Objekte rendern
3. Spieler Objekt rendern
4. Benutzeroberflaeche rendern

Diese Besonderheiten vereinfachen die Berechnung im 2D-Darstellungsfall. Dadurch ist dieser Fall aber nicht so gut optimiert, wie die 3D-Anwendung, und es werden regelmäßig Grenzen ausgereizt.

3.2 Benchmarking

Um eine möglichst einfache Bedienung zu gewährleisten, soll der Benchmark vollautomatisiert erfolgen. Wichtigste Eigenschaft des Benchmarks ist die Vergleichbarkeit der Ergebnisse. Dazu wurde die Renderzeit pro Frame bei X Objekten als Kennzahl eingeführt. Wie bei allen empirischen Untersuchungen muss diese den folgenden drei Kriterien genügen:

Validität Dass sich die Kennzahl mit empirischen Messergebnissen deckt, ist garantiert, da sie empirisch zustande kommt. Es wurde die in Entwicklerkreisen übliche Größe *Renderzeit pro Frame in Sekunden* gewählt, im Gegensatz zu ihrer Inversen, *Frames per Second*, oder einem arbiträren Punktesystem.

Objektivität ist durch die Wohldefiniertheit der einzelnen Testverfahren gegeben (siehe 5.2).

Reliabilität Als Maßnahme für hohe Reliabilität werden alle Messungen jeweils zehn mal durchgeführt und anschließend der Durchschnitt gebildet, um Einflüsse von Hintergrundprozessen zu minimieren. (siehe ..)

4 Konzept und Design

4.1 Allgemeines

Bei der Konzeption einer Benchmark gibt es grundsätzlich zwei verschiedene Herangehensweisen: Zum einen kann man eine repräsentative, in der Regel sehr komplexe Test-Szene rendern lassen, in der alle möglichen Schwierigkeiten und Bottlenecks vorkommen, die gemeinhin bei der Grafikprogrammierung auftauchen, und damit die GPU an die Grenze der Belastbarkeit zu bringen. Das ist der übliche Ansatz bei im Internet frei verfügbaren Benchmarks[1], die dazu dienen, die Fähigkeiten der eigenen Grafikkarte, also Hardware, zu testen. Zum anderen kann man viele einzelne, feingranulare Tests definieren, die auf einzelne Bottlenecks abzielen. Für das BunnySuite-Framework haben wir diese Herangehensweise gewählt, da das Framework explizit für den Vergleich von Grafik-Libraries konzipiert ist. Der Vorteil von feingranularen Tests ist, dass die Ergebnisse für die Entwickler sehr viel aussagekräftiger sind, weil sie ihnen präzise Hinweise geben, an welchen Stellen das eigene Framework hinter die anderen abfällt und wo eine Optimierung am meisten bringen würde. Außerdem vereinfacht dieser Ansatz die Entwicklung und Implementierung der Tests enorm. Ein Nachteil ist, dass die Tests teilweise artifizielle Anforderungen stellen, die in echten 2D-Spielen nicht vorkommen. Dieser Nachteil wird aber dadurch ausgeglichen, dass man die Tests – soweit sinnvoll – frei kombinieren kann. Außerdem ist die spätere Erweiterung des Frameworks um komplexere Tests durchaus möglich.

4.2 Identifizierte Bottlenecks

Da jede 2D-Grafik-Bibliothek ihre eigenen Stärken und Schwächen aufweist und für unterschiedliche Zwecke optimiert ist, ist es wichtig, differenzierte Tests durchzuführen, um gezielt mögliche Schwachstellen zu identifizieren, ohne dabei die Stärken zu ignorieren. Zu diesem Zweck wurden verschiedene "Bottlenecks" identifiziert, also häufige Schwachstellen, für die gezielt Tests entwickelt wurden. Diese sind unter anderem:

- Animieren der Objekte
- Objekte mit veränderlichen Texturen
- Vielzahl an gleichzeitig dargestellten Objekten mit unterschiedlichen Texturen
- Skalierung der Objekte
- Rotation der Objekte
- Objekte mit (halb-)transparenten Texturen

[1] Ein Beispiel: https://unigine.com/products/benchmarks/heaven/

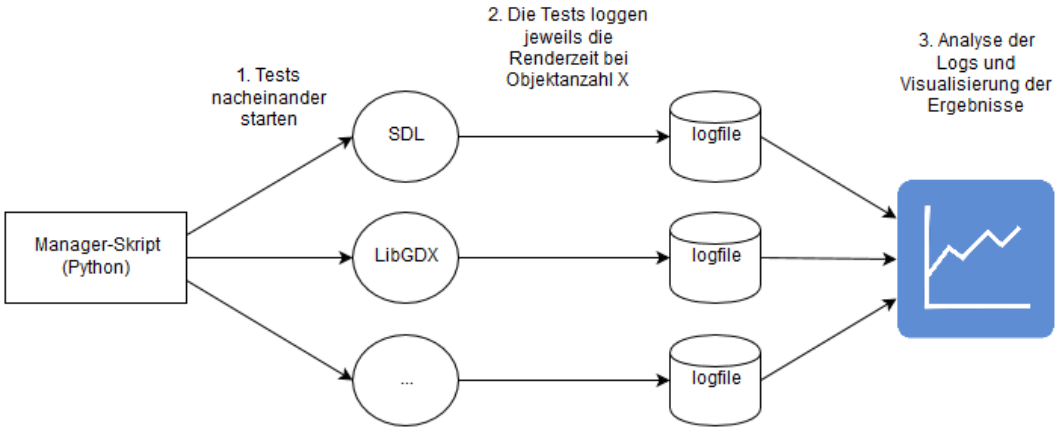
5 Umsetzung und Tests

5.1 Ablauf der Tests

Der Ablauf eines Tests im *BunnySuite*-Frameworks ist in Abbildung 5.1 schematisch dargestellt. Das Framework besteht im Wesentlichen aus zwei Teilen:

1. Das **Manager-Skript** ist dafür zuständig, den spezifizierten Test für alle Frameworks nacheinander zu starten, anschließend die Logs zu analysieren und die Ergebnisse in einem Diagramm zu visualisieren.
2. Es gibt eine **ausführbare EXE-Datei** für jedes Framework. Diese Datei wird vom Manager-Skript gestartet und bekommt die Parameter für die Messung als Argumente über die Command-Line übergeben. Das Programm führt die spezifizierten Tests durch und loggt die Renderzeit pro Frame bei X Objekten.

Figure 5.1: Ablauf eines Tests



Der auszuführende Test wird durch folgende Parameter spezifiziert:

test_name Der Name eines Tests. Die Tests werden in 5.2 aufgelistet.
min_val Startwert für die Anzahl X der zu zeichnenden Objekte
max_val Endwert für X
step Schrittweite, um die X nach jeder Messung erhöht wird

5.2 Beschreibung der Tests

Testname	Beschreibung
standard	Bunnies werden in die linke obere Ecke gezeichnet.
random	Bunnies werden in jedem Frame an eine neue zufällige Stelle gezeichnet.
scaled	Bunnies werden an zufällige feste Position gezeichnet. Mit jedem Frame wird die Skalierung um ein "Wachstum" verändert $initial 0.1$. Immer wenn die Skalierung $>= 5$ oder $<= 0.2$ ist, wird das Wachstum mit -1 multipliziert.
multitexture	Bunnies werden an zufällige feste Position gezeichnet. Jeder Bunny hat eine feste Textur, die zufällig aus drei Texturen ausgewählt wird.
texturechange	Bunnies werden an zufällige feste Position gezeichnet. Mit jedem Frame bekommt jeder Bunny eine zufällig aus drei Texturen ausgewählte neue Textur.
animation	Bunnies werden in die linke obere Ecke gezeichnet und mit zufälligen Geschwindigkeiten in x- und y-Richtung initialisiert. Mit jedem Frame wird die Position des Bunnies um die Geschwindigkeit verschoben. Zusätzlich wird zur y-Geschwindigkeit mit jedem Frame ein Gravitationswert in Richtung (0,-1) addiert. Verlässt der Bunny dadurch den Bildbereich nach links oder rechts, wird die x-Geschwindigkeit mit -1 multipliziert. Verlässt er den Bildbereich nach unten, wird die y-Geschwindigkeit mit -0.8 multipliziert und zusätzlich mit 50-prozentiger Wahrscheinlichkeit ein zufälliger Wert zwischen 3 und 7 addiert. Verlässt er den Bildbereich nach oben, wird die y-Geschwindigkeit auf 0 gesetzt. Dadurch entsteht der Eindruck einer Sprungbewegung.

6 Zusammenfassung und Ausblick

TODO