h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Lab Report

Master Projekt System Entwicklung, SS 2013
(*Prof. Dr. J. Wietzke, Prof. Dr. E. Hergenröther*)

„Was Sandkastenspiele mit der Frischluftzufuhr in Städten zu tun haben“

vorgelegt von

T. Sturm (709794)
A. Holike (724986)
S. Arthur (715720)
M. Djakow (718531)

01.05.2013

Inhaltsverzeichnis

1 Einleitung **2**

2 Bestehende Arbeiten **3**

3 Konzept **4**

4 Grundlagen **5**
4.1 Mathematische Verfahren 5

5 Realisierung **6**
5.1 Hardwareaufbau 6
5.2 XNA Game Studio 4.0 + Kinect SDK 7
5.3 Kinect Integration 8
5.4 Der Renderer 9
5.5 Das Kamerasystem 10
5.6 Terrain 11
5.7 Partikelsystem 12
5.8 Die Benutzeroberfläche 13

6 Zusammenfassung **14**

7 Probleme **15**
7.1 Echtzeitfähigkeit 15
7.2 Darstellung 15

8 Ausblick **15**

1 Einleitung

hier kommt einleitendes gequatsche

2 Bestehende Arbeiten

wie wurde das in den usa gemacht

[**Kreylos2010**]
was gibt es für ähnliche ansätze
beispiele für kinect
beispiele für xny
beispiele für partikelsystem

3 Konzept

wie sieht unser konzept aus

4 Grundlagen

Hier kommt immer die Kapitelüberschrift hin, ein kleines Vorgeplänkel was im Kapitel behandelt wird.

4.1 Mathematische Verfahren

4.1.1 Verzerrung von Bildern

4.1.2 Billboarding

Um unsere Vorgabe der Echtzeitfähigkeit zu erfüllen benötigt es ein paar Tricks, die es erlauben die Komplexität unseres Renderers zu minimieren, gleichzeitig jedoch darf dem Zuschauer diese Manipulation nicht bemerken. Eine beliebte Technik hierfür ist das Billboarding. Die Idee des Billboardings basiert darauf, komplexe geometrische 3D-Objekte auf ein zweidimensionales Rechteck das sogenannte Billboard runterzubrechen. Bei dem Billboard handelt es sich meist um ein vorher berechnetes Bild von dem ursprünglich darzustellenden 3D-Objekts. Anschließend wird dieses Billboard zur Kamera ausgerichtet, dem Zuschauer fällt es somit sehr schwer zu erkennen, das es sich bei dem gezeigten Objekt um eine zweidimensionale Kopie des 3D-Objektes handelt. Diese Technik wird hauptsächlich dazu verwendet die benötigten Rechenoperationen für Objekte welche in der Ferne liegen zu minimieren. Kommt die Kamera dem tatsächlichen Objekten sehr nahe, wird meist mit einer Interpolation zwischen dem Billboard und dem tatsächlichen 3D-Objekt umgeschaltet.

5 Realisierung

In diesem Kapitel wird die Realisierung des Projekts erläutert. Begonnen mit dem Hardwareaufbau über die Wahl der Frameworks, die Integration der *Kinect*-Kamera, dem Aufbau des Renderers, des Kamera-Systems und der Terrain-Darstellung bis zum Partikelsystem samt Physik.

5.1 Hardwareaufbau

Die Hardware
Die Konstruktion
Der Aufbau

5.2 XNA Game Studio 4.0 + Kinect SDK

Für die Interaktion mit der *Kinect*-Kamera und der Darstellung der Landschaft und des Partikelsystems haben wir uns für das *XNA Game Studio 4.0* und das Kinect SDK von *Microsoft* entschieden. Das *XNA Game Studio* ist eine Programmierumgebung die auf *Visual Studio* basiert und zur Entwicklung von Spielen für *Windows-Phone*, *XBox 360* und *Windows*-basierten Computern entworfen wurde. Bestandteil des *XNA Game Studio* ist das *XNA Framework*, welches mehrere auf dem *.Net-Framwork* basierende Bibliotheken vereint und eine sehr einfache und angenehme Schnittstelle zu diesen bereitstellt.

Dazu gehören:

DirectX
: DirectX ist eine API für hochperformante Multimedia-Anwendungen und kommt meist bei der Hardware-beschleunigten Darstellung von 2D- und 3D-Grafiken zum Einsatz.

XInput
: XInput ist eine API zur Verarbeitung von Benutzereingaben über Maus, Tastatur und den *XBox 360* Kontroller.

XACT
: XACT(Microsoft Cross-Platform Audio Creation Tool) stellt einfache Schnittstellen zur Audiowiedergabe und der Verknüpfung von Sounds an bestimmte Ereignissen bereit.

In unserer Implementierung wird ausschließlich DirectX für die Darstellung und XInput für die Verarbeitung der Benutzereingaben genutzt. Zudem kommt zusätzlich das *Kinect SDK* zur Ansteuerung der *Kinect*-Kamera zum Einsatz, welches auch auf dem *.Net-Framework* basiert und sich dadurch nahtlos und ohne weitere Anpassungen in das System integrieren lässt.

5.3 Kinect Integration

hier wird die dll erklärt und wie sie eingebunden wird
kinect baut metrik vom bild um veraenderungen wahrzunehmen
sendet event nur wenn neues Tiefenbild vorhanden
tiefenbild blur

5.4 Der Renderer

wie wird es erzeugt
einfärbung
höhenlinien

5.5 Das Kamerasystem

Um eine Navigation in unserer 3D Szene, sowie eine einfache Art der Kalibrierung zu ermöglichen, wurde ein kleines, erweiterbares Kamerasystem entwickelt. Das System besteht aus zwei Hauptkomponenten, der Kamera-Klasse und der Kamerakontroller-Klasse.

Kamera

Die Kamera-Klasse stellt alle Grundfunktionen einer virtuellen Kamera zur Verfügung. Dazu gehören neben der Translation und der Rotation auch unterschiedliche Arten der Projektion (Perspektivisch, Orthografisch) und verschiedene Kamera-Modi (Orbital, Walk, Fly) zur Navigation.

Um den sogenannten *Gimbal Lock* zu vermeiden, welcher bei der Verwendung von *Eulerwinkeln* zur Rotation entstehen kann und in speziellen Fällen den Verlust eines kompletten Freiheitsgrades bewirkt, setzten wir in unserem System auf den Einsatz von *Quantenionen* zur Rotation der Kamera. Diese bieten neben der Vermeidung des *Gimbal Locks* auch eine weitaus effizientere Berechnung der Transformationen.

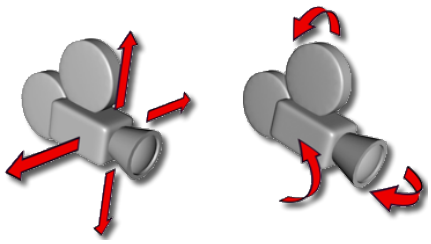

Abbildung 1: Grundfunktionen der Kamera.

Kamerakontroller

Die Kamerakonroller-Klasse dient als Schnittstelle zwischen der Peripherie und der Kamera und ermöglicht somit eine saubere Trennung zwischen der Verarbeitung von Benutzereingaben und der eigentlichen Funktionalität der Kamera. Abbildung 2 zeigt den groben Aufbau des Kamerasystems.

Abbildung 2: Aufbau des Kamerasystems.

5.6 Terrain

wie wird es erzeugt
einfärbung
höhenlinien

5.7 Partikelsystem

wie ist das partikelsystem gebaut
was kann das ding

5.8 Die Benutzeroberfläche

Nachdem die Grundfunktionalität der Hauptkomponenten unseres Systems standen ging es nun daran eine einfache aber dennoch funktionale Benutzeroberfläche zu entwerfen. Da das *XNA-Framework* von Haus aus auf *Windows-Forms* zur Darstellung von Benutzeroberflächen setzt, beschlossen auch wir vorerst diese Variante zu nutzen. Hielten uns aber die Möglichkeit offen eventuell später auf das etwas modernere *WPF*-System zu wechseln.

Hauptanforderungen waren ein übersichtliches Design und ein einfaches Hinzufügen von neuen Funktionalitäten. Um diese Anforderungen zu erfüllen entschieden wir uns für eine schlichte Statusleiste am unteren Rand des Editor-Fensters für einfache Anzeigen wie zum Beispiel die Frames Pro Sekunde(FPS) oder die Anzahl der Partikel und ein Tab-Panel an der rechten Seite des Editor-Fensters zur Konfiguration der einzelnen Komponenten. Durch die Nutzung des Tab-Panel lässt sich eine gute Separierung der einzelnen Komponenten in der Benutzeroberfläche realisieren.

Die Kommunikation zwischen der Benutzeroberfläche und den einzelnen Komponenten ist über das im *.Net-Framework* integrierte Event-System realisiert. Bei einer Interaktion mit der Benutzeroberfläche wird ein entsprechendes Event gefeuert, welches anschließend die benötigten Daten an alle Komponenten liefert, die sich zuvor für dieses Event registriert haben.

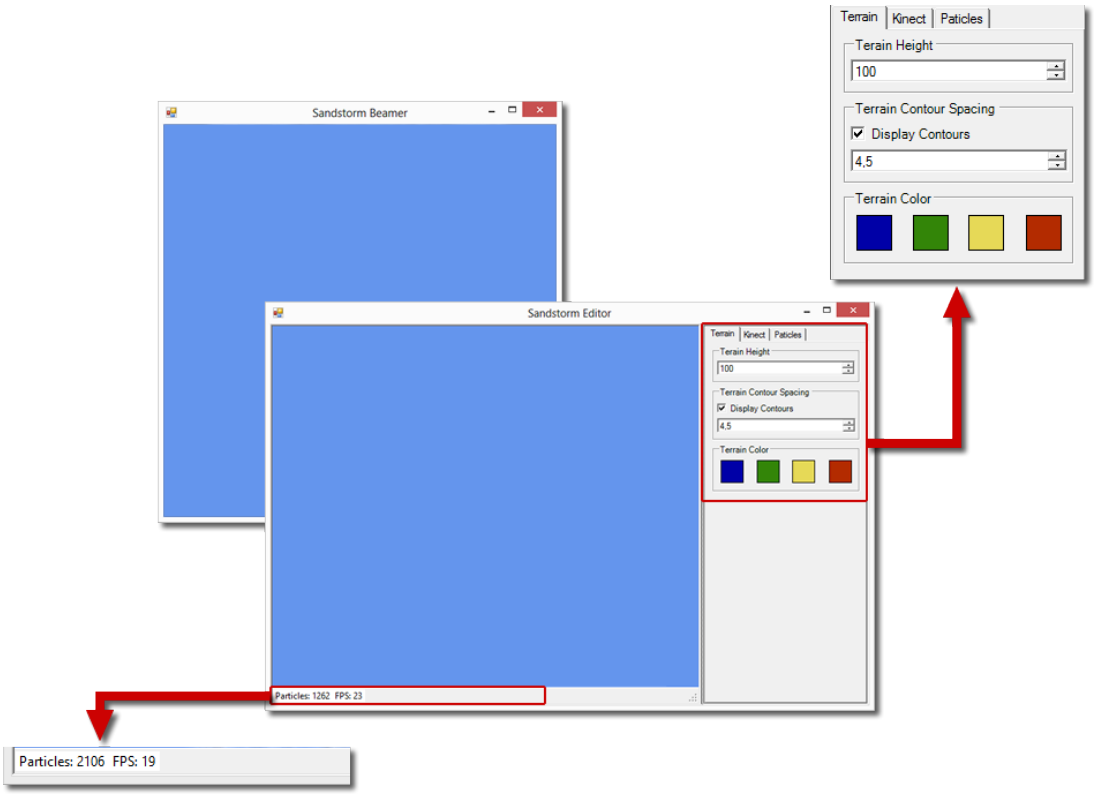

Abbildung 3: Die GUI

6 Zusammenfassung

hier schreiben wir unsere erfahrungen rein undwas wir genau hinbekommen haben. zudem sollen probleme die währed der arbeit aufgetreten sind erwähnt / erläutert werden.

7 Probleme

7.1 Echtzeitfähigkeit

Leider besitzt die derzeitige Ausarbeitung diverse kleinere Probleme, welche die Echtzeitfähigkeit des Systems gefährden. Diverse teile von Berechnungen werden noch wie in **??** beschrieben auf der CPU ausgeführt, während der Teil der Visualisierung bereits auf die GPU portiert wurde. Dies führt zu erheblichen Performanceproblemen, denn es muss bei jeder Physikberechnung (jeden Frame), die Partikeldaten zwischen GPU und CPU kopiert und synchronisiert werden.

7.2 Darstellung

Die Darstellung stellte sich um Laufe des Projektes als schwieriger heraus als vorher angedacht. Hierbei kann man die Probleme auf welche wir gestoßen sind grob in Hard- und Softwareprobleme unterscheiden.

7.2.1 Hardware

Trotz das wir einen Beamer von einem Grafiklabor der Hochschule zur Verfügung gestellt bekommen haben, bemerkten wir bereits bei ersten Tests, das ein großer Farbunterschied zwischen Beamer und Monitor vorhanden ist. Leider scheint das Spektrum unseres Beamers sehr begrenzt zu sein, so das wir einen Farbunterschied zwischen weiß und gelb kaum wahrnehmen können.

7.2.2 Software

Durch die physikalische Gegebenheit das Kinekt und Beamer sich an unterschiedlichen Orten befinden, entsteht bei der Projektion zusätzlich zur Verzerrung auch noch das Problem der Verschiebung. Die Kalibrierung stellte sich somit schwieriger heraus als bisher gedacht, deshalb wurden aus zeitlichen Gründen der Fokus auf Aufgaben gesetzt um schnellstmöglich eine lauffähige Version zu erstellen.

8 Ausblick

Trotz das auf uns allerlei Probleme zukamen, entstand im Laufe eines Semesters eine Echtzeit Sandkastensimulation, die bereits grundlegende Funktionalität bietet. Im Laufe des nächsten Semesters werden wir dann Aufgaben, welche in diesem Semester ein wenig vernachlässigt wurden wie z. B. die Kalibrierung nachbessern. Des Weiteren werden wir die bisherigen Physikberechnungen auf die GPU portieren um so hoffentlich wieder die Echtzeitfähigkeit des Sy-

stems zu erlangen. Auch neue Funktionalitäten sind geplant, welche notwendig sind um unser eigentliches **??** zu erreichen.