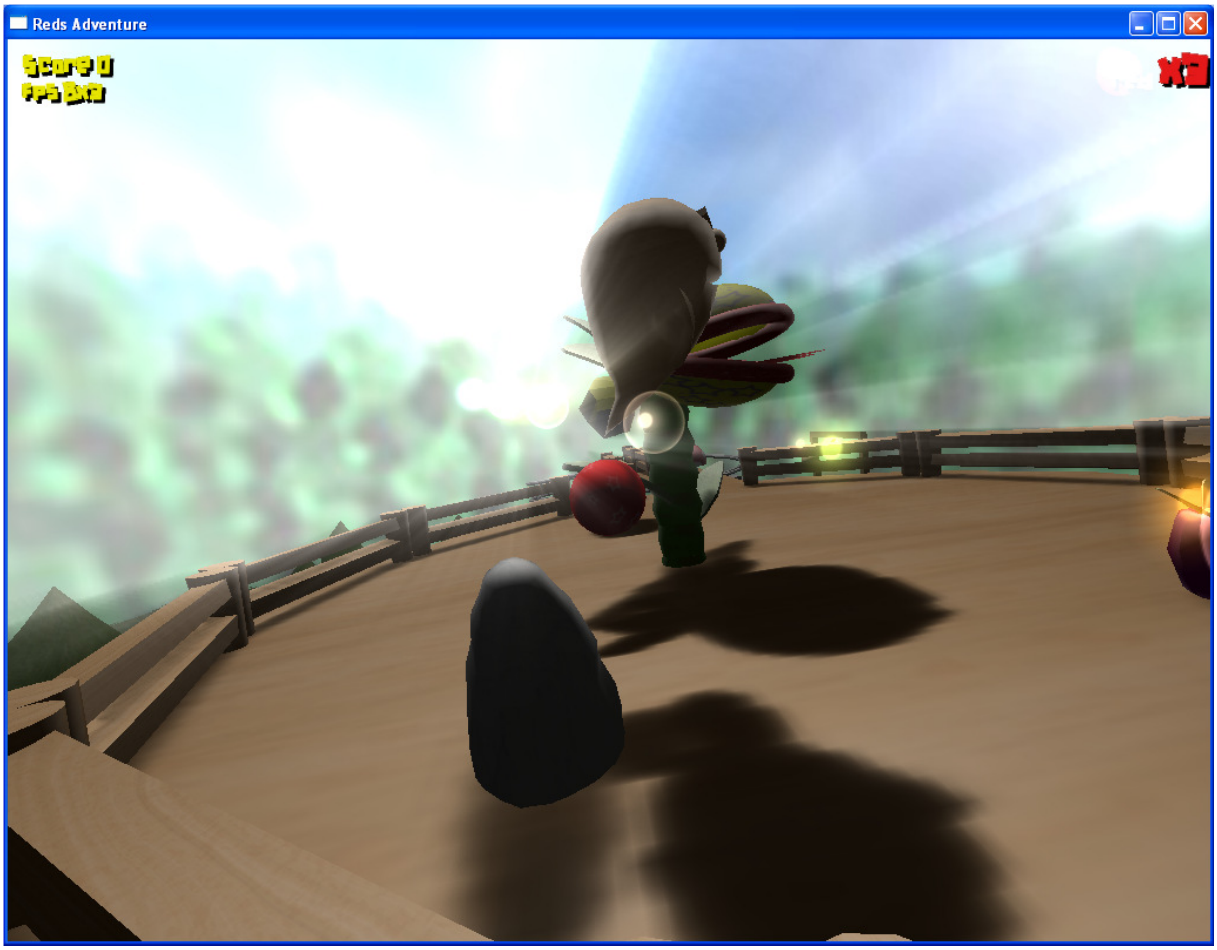
Red's Adventure – 3. Abgabe *– CG2/3 LU*

Michael Beham, 0726417, 532, e0726417@student.tuwien.ac.at

Johannes Sorger, 0225843, 932, johannessorger@gmx.net



Gameplay

Red's Adventure ist ein Arcade-Geschicklichkeitsspiel, das aus einer Kombination des Spielprinzips von „Kugel-Labyrinth" und Pac-Man besteht. Das Ziel des Spiels ist es eine Kugel durch ein Labyrinth zu führen, dabei diversen Hindernissen und Gegnern ausweichen und Items einzusammeln, um Powerups und Punkte zu erhalten. Begonnen wird an einer fix vorgegebenen Startposition. Der Spieler muss versuchen die Kugel sicher in den Zielbereich zu manövrieren. Am Weg zum Ziel lauern zahlreiche Gefahren, wie Abgründe und Gegner, denen mit Geschick ausgewichen werden muss. Außerdem müssen Items eingesammelt werden, um das Ziel mit einer möglichst hohen Punktezahl zu erreichen und die Highscore zu knacken.

Gesteuert wird nicht die Kugel direkt sondern die Welt, in der sie sich befindet. Durch Bewegung des Mauscursors oder Kippen/Neigen der WiiMote kippt man die Welt in eine bestimmte Richtung und die Kugel rollt entsprechend der Neigung.

In jedem Level gibt es eine Kanone, die das Ziel repräsentiert. Durch Rollen der Kugel in die Mündung der Kanone wird der Spieler zum nächsten Level befördert. Auf dem Weg dahin lauern verschiedene Gegnertypen, wie zum Beispiel Geister, die den Spieler hartnäckig verfolgen. Über jedes Level ist auch eine Vielzahl an Items verstreut: Sterne, die eingesammelt werden müssen, um eine möglichst hohe Punktezahl zu erreichen; Beeren, erhöhen die Anzahl der verbleibenden Leben/Versuche um 1.

Nichttriviale Objekte

Wir modellieren wie im Punkt Tools zum Modellieren die Objekte mit Maya. Anschließend werden sie mithilfe des OgreExporters exportiert und mit dem OgreXMLConverter von den OgreCommandLineTools in ein XML Format konvertiert. Im Spiel werden diese dann mithilfe von tinyXML geladen.

Animierte Objekte

Einige Objekte bewegen sich im Spiel. Wir benützen um Animation darzustellen Vertex Animationen. Der Code ist genauso wie im Repititorium vorgestellt. Dem Shader werden immer 2 Keyframes übergeben. Ein Keyframe wird dargestellt mit glVertexf(..) bzw. glNormal(...) und der andere Keyframe wird durch die Texturkoordinate 5 (Vertices) und 6 (Normals) übergeben. Im Vertex Shader wird mithilfe der der mix() Funktion linear interpoliert anhand einer Gewichtung.

Wichtig ist aber noch, dass wir 3x die Vertices und Normals animieren. Einmal für die Speicherung der Tiefenwerte für die Schattierung, einmal für die Darstellung der Animation und einmal die Berechnung des FBO Bildes für Light Scattering.

Transparenz-Effekte

Weiters haben wir auch transparente Objekte benützt. Sie werden wie in den Repititoriumsfolien beschrieben am Ende gezeichnet. Außerdem wird die Dephmask beim Zeichnen deaktiviert und erst nachher wieder aktiviert.

Steuerung:

Mausbewegung: Bewegung des Spielfeldes

Linke Maustaste + Mausbewegung: Zoom in/out bzw drehen nach links oder rechts

Wii remote kippen: Bewegung des Spielfeldes

Wii remote Cursertasten: Zoom in/out bzw drehen nach links oder rechts

F1: Hilfe anzeigen

F2: Framerate anzeigen

F3: Wireframe anzeigen

F4: Texturefiltermethode ändern (nearest, linear)

F5: Mipmapmethode ändern (ausgeschalten, nearest, linear)

F6: Rendermethode ändern (immediate, vertex array, vertex buffer object)

F7: Verwendung von Displaylisten

F8: View frustum culling ein/aus

F9: Transparenz ein/aus

F10: BoundSpheres für View- Frustum- Culling

F11: Darstellung der Shadow Map

M: Einschalten des Wii Remote Controllers

S: Shadow Maps ein/aus

L: Light Scattering ein/aus

Z: Zu Anfangspunkt

N: Zur Kanone

Effekte

Level of Detail for static mesh
motion blur
phong shading
shadow mapping (variance shadow mapping)
radial blur
light scattering
lens flare effekt
particle effect

Genauere Beschreibung der Effekte

Level of Detail for static Mesh

Dieser Effekt wählt anhand der euklidischen Distanz zwischen dem Objekt und dem Kamerapunkt die Qualität des Objektes aus. Dazu haben wir das Objekt in verschiedensten Qualitätsstufen abgespeichert. Die Qualitätsstufe des Objekts wird ausgewählt, indem die Länge zwischen View- und Farplane durch die Anzahl der gespeicherten Qualitätsstufen dividiert wird und dann mit der euklidischen Distanz zwischen Kamera- und Objektpunkt verglichen.

Sichtbar ist dieser Effekt im 1. Level bei den Tannenbäumen. Alle Bäume sind gleich, aber sie sehen verschieden aus, da 5 Level of Detail Stufen benützt werden. Die näheren sehen besser aus, die weiter weg sind weniger gut. Wir haben die Unterschiede sehr groß gewählt, damit man sehr gut das arbeiten des Level od Detail sieht.

Da ich keine Informationen zu diesem Effekt gefunden haben, wurde er nach Verständnis implementiert.

Motion Blur

Motion Blur wurde von uns auch sehr einfach implementiert. Grundsätzlich haben wir uns an die Accum Buffer Implementierung wie im Red Book gehalten. Anstatt aber den Accum Buffer zu benützen, aber wir es mit einem einfachen Fragment Shader implementiert. Es wird einfach jedes Pixel des vorherigen Rendering Durchgang mit dem Pixel des aktuellen Rendering Durchgang multipliziert. Die jeweiligen Renderingdurchgänge werden in einem Framebufferobjekt gespeichert.

Der Effekt wird aktiviert, wenn man durch eine Kanone in das nächste Level geschossen wird, oder wenn man die linke Maustaste drückt.

Referenz:

Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis: „Open GL Programming Guide - Kapitel 10 The Framebuffer – The Accumulation Buffer“

Phong Shading

Als weiteren Effekt haben wir auch Phong Shading implementiert. Anstatt das Shading von Open GL (Ground Shading) zu benützen, haben wir das Phong Shading benützt. Den Farbwert berechnen wir uns im Fragmentshader (Pixelshader) und daher bekommen wir einen schöneren Farbwert.

Im Vertex Shader berechnen wir uns den Normalvektor, einen Vektor der Richtung des Lichtvektors und des Auges. Weiters berechnen wir uns noch das Globale Ambiente Licht, indem wir das Globale Ambiente Licht mit dem ambienten Material multiplizieren. Außerdem berechnen wir auch die euklidische Distanz zwischen den Vertex Punkt und dem Lichtpunkt.

Im Fragment Shader berechnen wir uns die Farbe des Pixels. Ich berechne dazu zuerst einmal einen Abschwächungswert att. Als nächstes berechne ich das ambiente Lichtanteil, diffusen Lichtanteil und spekulären Lichtanteil. Dieses wird addiert und wir haben den korrekten Beleuchtungswert.

Den Effekt sieht man auf der ganzen Szene. Ohne Schattierung sieht man es, wenn man die Taste S drückt bzw. bei den Hintergrund Objekten (Tannen, Boden) (hier ist es immer ausgeschaltet).

Referenz:
http://www.lighthouse3d.com/opengl/glsl/index.php?pointlight
http://wiki.delphigl.com/index.php/Tutorial_glsl2
Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis: „Open GL Programming Guide - Kapitel 5 Lighting“

Radialblur

Als weiteren Effekt haben wir Radialblur eingebaut. Von der Mitte der Szene wird das Bild geblurt. Dazu wird das Bild in einem FBO gespeichert und auf ein Quad projeziert. Anschließend wird vom Mittelpunkt aus wird das Bild verblurt (16 Samples). Die Samples werden addiert, anschließend durch 16 dividiert und mit einem Helligkeitsfaktor addiert.

Shadow Mapping

Für die Schatten haben wir Shadow Mapping eingesetzt. Genau genommen, müssen wir von Variance Shadow Mapping sprechen. In einem ersten Renderingdurchgang wird die Szene von Punkt des Lichtes gerendert. Beim Rendern der Szene wird die Tiefe und Tiefe zum Quadrat gespeichert in einem FBO.

Anschließend werden die Bias Matrix multipliziert. Anstatt aber die Inverse Projektionsmatrix und Modelviewmatrix zu berechnen (wie im Repititorium vorgeschlagen), habe ich einfach die beiden Matrizen gespeichert. Anschließend eine neue Identititätsmatrix erstellt, die Bias Matrix multipliziert und dann die beiden Anderen. Nun wurden die Schatten vertikal und horizontal verblurt. Dafür benützen wir einen Gauß Shader mit 7 Samples für jedes Vertex.

Als nächstes wird die Szene gezeichnet von dem Kamerapunkt. Hier wird dann neben dem Phong Shading auch die Schattierung benützt. Dazu benützen wir die chebyshev Formel, welche uns die Prozent der Schattierung berechnet. Beim Phong Shading wird dann die erhaltene Farbe einfach zusätzlichen Wert hinzugefügt.

Weiters muss noch gesagt werden, dass ich diesen Effekt (wie alle anderen) nicht unter eine Nvidia Karte programmieren konnte. Einerseits besitzen wir nur aktuelle ATI Karten, anderseits können wir im Studentenlabor nichts kompelieren, da man die Wii Remote Libary für jeden PC kompelieren muss, wo man programmieren will. Daher haben wir nun auch eine Taste S eingebaut, mit dem man die Lightmap ausschalten kann.

Referenzen:

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch08.html

http://www.punkuser.net/vsm/

http://fabiensanglard.net/shadowmappingVSM/index.php

Light Scattering

Bei diesem Effekt wird die komplette Szene am Anfang gezeichnet ohne der Skybox, dafür mit einer weißen Kugel (die Sonne). Die Hintergrundfarbe ist grau, das Licht, Textur und die Effekte sind ausgeschalten. Das Endbild wird in einem FBO gespeichert.

Anschließend wird die komplette Szene gezeichnet (mit Schatten, Transparenz, Lens Flare, usw.) und am Ende das im Frame Buffer gespeicherte Bild über die Szene geblendet. Beim blenden des Bildes über die Szene wird aber noch ein Shader aktiviert, der anhand des Punktes der Sonne die Szene verblurt.

Der Punkt der Sonne wird mit der Funktion gluProject() berechnet. Diese Funktion wird auch benützt um beim Lens Flare Effekt die Sichtbarkeit zu überprüfen.

Referenz:

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch13.html
http://fabiensanglard.net/lightScattering/index.php

Lens Flare Effekt

Auch diesen Effekt haben wir implementiert. Dazu wird am Schluss der Szene (zwischen Transparenten Objekte und Partikelsystem) der Lens Flare Effekt dargestellt.

Referenzen:

http://nehe.gamedev.net/

Wir haben das Grundgerüst (Texturen und Camera-oriented-Billboard) von hier. Wir haben jedoch den Rest abgeändert da wir mit anderen Vektorklassen und Kamerarepräsentation arbeiten.

Particle Effekt

Der Partikeleffekt wird jedes Mal aufgerufen wenn ein Stern eingesammelt wurde. Der Effekt wird verwendet, um ein Explodieren oder Zerplatzen des Sterns zu simulieren. Orientiert haben wir uns am Partikeleffekt aus dem Nehe Tutorial (siehe Quelllink). Der Effet wurde in der RParticle Klasse implementiert. Jedes Rparticle Objekt enthält ein Struct Particle, das einen einzelnen Partikel repräsentiert. Jeder Partikel hat Werte für Lebensdauer (Alphawert der Textur), Farbe, Position, Richtung der Fluglaufbahn und Schwerkraft (die den Partikel in eine bestimmte Richtung zieht). Der Effekt selbst besteht aus einem Array von 1000 dieser einzelnen Partikel. Jeder Partikel wird beim Erstellen des Rparticle Objekts mit bestimmten Werten initialisiert. Die einzelnen Partikel werden mittels Trianglestrips dargestellt. Die Partikel werden gezeichnet, indem jedes Frame eine Methode im Rparticle Objekt aufgerufen wird (`DrawGLScene()`). In dieser Methode wird eine Schleife aufgerufen, die die Werte aller 1000 Partikel ausliest, zeichnet und updated (Position, Geschwindigkeit, Alphawert). Wenn ein Partikel einen Alphawert von 0 erreicht hat, gilt er als erloschen. Sobald alle 1000 Partikel erloschen sind gilt der Effekt als beendet – die Variable ‚terminated', die den Zustand des Effekt bescreibt wird auf true gesetzt.

Die Partikel Objekte werden in der TrigggerReport.h in einen static Vektor geladen, sobald ein Stern berührt wurde. In der drawScene() Methode in physx2.cpp wird der Vektor jedes Frame ausgelesen und die DrawGLScene() Methode der sich darin befindenden Partikel wird aufgerufen.

Referenzen:

http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19

sonstige Besonderheiten

Wir können nicht nur mit der Maus das Spiel steuern, sondern auch mit der Wii Remote. Dafür braucht man einfach ein Bluetooth auf dem PC und eine Wii Remote.

Interessant an diesem Spiel ist sicher auch, dass wir die Kamera 2x rendern und die Matrix abspeichern. Einmal für den Hintergrund und einmal für das Spielfeld samt Licht. Für Hintergrundobjekte benützen wir dann einfach die Kamera für den Hintergrund und für das Spielfeld die Kamera für das Spielfeld. Das wird auch nötig, da wir in unserem Spiel nicht das Spielfeld drehen, sondern nur die Gravitationskraft ändern und die Kamera drehen. Beim Hintergrund wird die Kamera nicht gedreht, beim Spielfeld wird die Kamera gedreht. Dieses Prinzip haben wir uns von der Berechnung der Schatten abgeschaut, da andere Varianten nicht so gute Ergebnisse gebracht haben (leichtes springen usw.).

Zusatztools

GLSL: Pixel- und Vertexshader für Spezialeffekte (Schatten, Phongshading, usw.)

FMOD: Die Soundausgabe wurde mit FMOD gemacht.

DevIL: Zum Laden der Bilder (JPG, BMP, TIF)

Ogre Maya Exporter: Um das Mesh von Maya in eine .mesh Datei zu speichern

OgreXMLConverter: Umwandlung eines .mesh Files zu einer .xml File

Nvidias PhysX: Für physikalische Effekte, Kollisionserkennung, usw.

Wii YourSelf!: Für die Steuerung mit der Wii Remote

TinyXML: Zum Laden einer XML Datei.

GLFW: Windowmanager

GLEW: Zum Laden von Extensions

Tools zum Modellieren

Wir modellierten die Modelle mit Maya. Exportiert wurden sie mit Ogre Maya Exporter. Alle Objekte worden von uns selbstständig modelliert.

Interaktionsfolgen

Wie bereits erwähnt, wird das Spiel mit der Maus oder WiiMote gesteuert. Das Bewegen der Maus über das Spielfenster oder das Kippen/Neigen der WiiMote kippt/neigt die Spielwelt und die Kugel rollt in die dementsprechende Richtung.

Es gibt auch einige Tastaturcommands. Die Tasten F1 bis F9 haben die in der Aufgabenstellung verlangten Funktionen. Mit “S“ schaltet man Shadow Maps aus und ein, mit „L“ wird der Light Scattering Effekt aus und eingeschaltet.

Es gibt auch ein paar Cheat Tasten, die das Testen bzw. Präsentieren erleichtern. „Z“ resetted die Kugel an die Startposition des Levels, „N“ bringt den Spieler direkt zur Kanone, die einen ins nächste Level befördert.

Um in das nächste Level gelangen zu können, muss die Kugel in die Mündung der Kanone gerollt werden.

● *Links:*

[1] Ogre Maya Exporter: http://www.ogre3d.org/

[2] Ogre Commandline Tools: http://www.ogre3d.org/

[3] DevIL: http://openil.sourceforge.net/

[4] FMOD Sound System: http://www.fmod.org

[5] TinyXML: http://www.grinninglizard.com/tinyxml/

[6] Nvidia PhysX: http://www.nvidia.com

[7] Wii YourSelf!: http://wiiyourself.gl.tter.org/

[8] GLFW: http://glfw.sourceforge.net/

[9] GLEW: http://glew.sourceforge.net/