CG2 Abgabe: „Bubbleking“

Gerald Peter, 0525425 E532

Stephan Rotheneder ,0625931 E532

Das Spiel startet nachdem eine beliebige Taste gedrückt wird. Levels kann man während dem Spiel mit 1-4 auswählen. Ein Level kann mit r neugestartet werden. Die Steuerung erfolgt mit wasd , Maus sowie den up/down Tasten für die Schussstärke.

Kurzbeschreibung der Umsetzung der Anforderungen

Die F-Tasten Belegung wurde wie verlang umgesetzt. Zusätzlich wurden die F10 und F11 Tasten belegt. F10 schaltet zwischen den „Terrain as single object“ und „split up Terrain“ Modus um. Der Vorteil des Single Object Modus ist, dass das Terrain als Triangle Strip und als Buffer Object bzw. Display List gut gerendert werden kann. Beim Split up Modus wird das Terrain in einzelne Dreiecke aufgeteilt um das View Frustum culling anwenden zu können.

F11 aktiviert den Cartoon Shading Modus.

Folgende **Spezialeffekte** wurden implementiert:

- Partikelsysteme
- Komplexe Transparenz (von Partikeln beim Einsammeln von Münzen)
- Cartoon Shading mit 1D-Textur (mit F11 aktivieren!)

Partikelsysteme
Die Implementierung der Partikel wurde an das Nehe Tutorial bzw. an das Video tutorial angelehnt. Im Unterschied zu den Tutorials wurde auch jeder Partikel in die Physikengine miteinbezogen und als Gegenstand mit ausgeschalteter Kollisionserkennung versehen. Die Initialisierung des Geschwindigkeitsvektor des Partikels erfolgt durch einen Referenzvektor (beim Schuss des v Vektors des Schusses beim Aufprall) addiert mit einem zufälligen Vektor, damit die Partikel gestreut werden. Bei jedem Partikel wird die Transparenz von 0 auf 1 mit der Zeit interpoliert. Alle Partikel sind mit einer Textur versehen die einen radialen Farbübergang von schwarz auf weiß (Zentrum) darstellt. Das Partikel wird als Quad gezeichnet dass immer zur Kamera ausgerichtet ist (Normalvektor des Quads schaut Richtung Kamera).

Komplexe Transparenz
Transparente Objekte wurden nach der Entfernung zur Kamera sortiert. Dabei wird für den Distanzvergleich einfach auf den Z-Referenz Vektor der Kamera projiziert.

Cartoon-Shading
Eine simple 1D-Textur wird dem in GLSL geschriebenen Shadern übergeben. Diese besteht aus Schwellwerten und Beleuchtungswerten. Im Fragmentshader wird der entsprechende Beleuchtungswert errechnet, erreicht er einen der Schwellwerte wird dem jeweilige Pixel der Beleuchtungswert dieses Schwellwertes zugeteilt.

Besonderheiten
3ds loader, Character Controler für Kamera Kollisionen,Animierte Objekte wurden in Form von Clothes umgesetzt. Textile verformbare Bänder die mit der Umgebung interagieren. Sie verhalten sich ähnlich wie Gummibänder.

Libraries
Es wurden außer GLUT und der PhysX Engine keine Zusatzlibraries verwendet.

Models
Die 3ds Models stammen aus dem Internet bzw. wurden auch manuell nachbearbeitet.

Kurzbeschreibung der Umsetzung der Angabe

- Grundlegendes Gameplay
  Im Spiel Bubbleking kann man sich durch vier verschiedene Levels frei in drei Dimensionen bewegen .
  In der Map findet man Münzen und Kugeln die man abschießen kann.
  Wenn Münzen mit Kugeln oder Würfel kollidieren werden die Münzen eingesammelt und der Score steigt.
  Der Wert der Münzen ist abhängig davon wieviel Zeit schon seit Start des Levels verstrichen ist. Das Spiel endet nach 500 Zeiteinheiten.
- Bewegte Objekte
  Bewegte Objekte sind: die Münzen, die Kugeln und ein paar Würfel.
  Die Bewegung/Kollision selbst wird wird über die PhysikEngine PhysiX berechnet.
- Texturierte Objekte
  Texturen kann man finden den beweglichen Münzobjekten sowie an der Skybox und einer statischen Hütte auf dem Terrain.
- Einfache Beleuchtung
  Beleuchtung erfolgt über eine Lichtquelle von oben.
- Frei bewegliche Kamera
  Der Spieler selbst ist die Kamera und er kann sich frei bewegen.
- Funktionierende Steuerung
  Die Steuerung funktioniert über WASD + Maus.
  Weiters kann mit den Pfeil-Up und -Down Tasten die Schussstärke reguliert werden.

Kurzbeschreibung der Implementierung

Alle Objekte werden durch die Klasse object repräsentiert. Die Klasse object enthält Positionsangaben, Information über die Art des Objektes (Pickup, Dynamic,Static, Shoot, Clothes, Score, Terrain,..) und ein Objekt des Typs ObjetData, die die komplette Geometrie- und Renderinformation des Objektes enthält. Somit besitzt jedes "object" auch ein Objekt der Klasse "objectData". Die Klasse LevelData enthält alle Objekte in Form eines std::vector.
Ein Leveldata Objekt enthält somit alle Daten um ein komplettes Level zu laden. Zum Laden der Objekte haben wir ein Konzept einer erweiterten Heightmap entwickelt, bei dem aus einem Farbbitmap File alle Daten enthalten sind um daraus das Level zu generieren. Dabei stellt der Rot Wert des Bildes die Höhe des Terrains dar, der Grün Wert steht für einen Object Code (d.h. wir unterstützen 255 verschiedene Objekte pro Level wobei 251-255 für Standard Objekte reserviert sind) . Der Blau Wert ist eine optionale Höhenangabe für das Objekt(Bei Blau=0 wird das Objekt an die Höhe des Terrains gesetzt). Da in unserem Spiel keine komplexen Levelstrukturen vorgesehen sind, kann die recht eingeschränkte Flexibilität, jedoch die erhöhte Effektivität dieser Methode eingesetzt werden. Der Leveleditor ist somit praktischerweise ein gewöhnliches Grafikprogramm (zb Photoshop, gimp,...)

Die Klasse ObjectData stellt unterschiedliche Ausprägungen zur Verfügung um Renderdaten unterschiedlicher Objekte zu repräsentieren: Kugel, Würfel, Vertice Objekte inkl. 3ds ModelLoader, Skybox, Terrain,..

Nachdem alle Objekte über die Methode loadLevelFromBMP instanziert und in levelData Objekt abgespeichert wurden, werden diese der Klasse Physik übergeben. Es wurde die Aeiga PhysX Engine verwendet. Es wird je nach Objekttyp die entsprechende Geometrieinformation der Physikengine zur Physiksimulation übergeben. Dabei wird bei jedem Physik-Aktor ein Zeiger auf unser "object" übergeben.

In der Klasse Renderer werden alle Objekte gerendert (alle OpenGL Befehle getätigt). Dazu werden in der renderScene() Methode alle Physik-Aktoren übergeben um die aktuelle Position des jeweiligen Aktors zu bekommen, sowie die Renderinformation aus objectData.

Folgende Features wurden implementiert (die Tutorials/Quellen an denen gute Erkenntnisse gewonnen wurden stehen in [...] dabei):

- Physik Simulation für Kugeln (mit CCD Skelleton für Schussobjekte), Würfel, heightfield, TriangleMeshes und convexMeshes, Clothes,[PhysX Dok.]
- View Frustum Culling - Radar Approach (ein/ausschalten mit f) [LightHouse 3D]
- Frei bewegliche Kamera mit CharacterController für die Kollisionserkennung (awsd und Maus) [RedBook und PhysX]
- 3ds models mit Texturen laden und rendern (Pickups und statische Objekte) [http://www.spacesimulator.net/tut4_3dsloader.html]
- Imageloader für bmp [www.gamedev.net Forum]
- Terrain/Heightmap (http://www.videotutorialsrock.com und PhysX]
- zusätzlich Leveldaten über Farbkanäle der Heightmap einlesen
- Lichtquelle und amb. dif. und spec. Reflection [RedBook]
- Gouraud Shading über die Fixed Pipeline[LightHouse 3D]
- Cartoon Shading mit 1D-Textur (aktivieren mit F11) [LightHouse 3d]
- 3DS-Loader zum Laden der 3D-Modelle[spacesimulator.net]
- Skybox [http://ohad.visual-i.com/exper/exper.htm]

- Partikelsysteme [nehe und http://www.videotutorialsrock.com]

- Komplexe Transparenz [http://www.videotutorialsrock.com und aus eigenen Überlegungen]

Texturen

Weiters wird in der ObjectData-Klasse gegebenenfalls die Textur des Objekts geladen, kreiert und gebunden und die Texturkooridinaten werden zudem generiert bzw. aus dem 3ds-Model gelesen. Die Texturen werden mit der Klasse ImageLoader erzeugt. Diese stellt eine statische

Methode loadBMP(const char *Filename) zur Verfügung, welche die Pixelwerte aus 24Bit unkomprimierten BMP-Files lesen kann. Diese werden in einer Image-Klasse mit Längen- und Breiteninformation verpackt und zurückgegeben.
Dieses Image kann dann einer weiteren statischen Methode des ImageLoaders übergeben werden und zwar loadTexture(Image* image) oder es kann wie oben bereits beschrieben für die Mapgenerierung verwendet werden.
LoadTexture lädt die Pixelwerte in den Texturspeicher und liefert die entsprechende TexturID zurück die dann immer wieder mit glBindTextur(GL_TEXTURE_2D, TextureID) gebunden werden kann.

Shader

Wir haben unsere eigenen Shader implementiert zwischen denen mit Hilfe der F11-Taste hin und her gewechselt werden kann. Im Vertexshader wird die Farbe für die Eckpunkte eines Faces berechnet und über eine varying Variable an den Fragmentshader weiter gegeben. Außerdem werden die Transformationen der Vertices durchgeführt, hierzu wird allerdings die Fixed-Pipeline verwendet.
Im Fragmentshader wird die Farbe eines jeden Pixels bestimmt und festgelegt.
Primär wird hierzu mit Hilfe eines 2D-Samplers die Textur geladen und als Farbe definiert, gibt es keine Textur so wird die Farbe des Objektes (das Material) verwendet.
Im Falle der Partikelsysteme wird beides verwendet.
Objekte wie Text und Fadenkreuz durften nicht gelighted werden, hierfür steht ein entsprechendes Bit im Shader zur Verfügung, mit dem man einzelne Objekte vom Licht unabhängig zu shadern.
Ist nun noch das Bit für das Cartoonshading gesetzt, erfolgt noch eine Multiplikation mit einem Wert aus der 1D-Textur. Diese 1D-Textur des Cartoonshaders wird über 2 Felder bestimmt, die in der Methode RenderScene() des Renderes gesetzt werden. Die beiden Felder sind ein Tresholdfeld und ein Valuefeld, deren erste n-1 Werte sind jeweils als Paar zusammen zu fassen. Für den letzten Wert des Value-Feldes gibt es keine Entsprechung, da dieser Wert als Unterschranke gewertet wird. Die Berechnung erfolgt so dass der Winkel zwischen Pixelnormale und Lichtquellenrichtung berechnet wird (Dotproduct!) und je nachdem welcher Schwellwert überschritten wird, wird eine andere Beleuchtungssituation herbeigeführt.

3DS-Loader

Unser Spiel verfügt auch über einen simplen 3DS-Loader.
Die ursprüngliche Idee des 3DS-Fileformats ist der aufbau eines Baumes. Diese Strucktur haben wir zu Anfang versucht in Code abzubilden, diese Idee wurde aber verworfen, da ein 3DS-File auch linear ausgelesen werden kann, wenn gewisse Richtlinien eingehalten werden. Darum werden in unserem Loader nur die obersten Ebenen des Chunk-Trees ausgelesen, bis die benötigten Daten (Vertexkoordinaten, Texturkoordinaten, etc.) gelesen wurden.
Danach wird der Lesevorgang abgebrochen.
Um diesen Loader in die Klassenstrucktur einzufügen haben wir die Klasse VerticeData um einen entsprechenden Konstrucktor erweitert. Somit sind wir in der Lage die gelesenen Daten in aufbereiteter Form dem Renderer zur Verfügung zu stellen.