PieWars Reloaded

Submission 2

Effekte

Die Engine von Pie Wars Reloaded beinhaltet folgende Effekte:

- **Bloom / Glow (1 Punkt):**

Wo man es bemerkt:
Unser Bloom-Effekt erscheint bei den Zielscheiben. Die weißen Ringe jeder Zielscheibe leuchten, damit man man sofort sieht, dass man diese treffen muss.

Implementierung:
Zuerst wird die ganze Szene in eine Textur gerendert mittels eines FrameBufferObjects (FBO). Dann wird die ganze Szene nochmals mit einem “GlowShader” in eine andere Textur, die Glowmap, gerendert. Die weißen Teile der Zielscheibe werden dabei als weiß in die Glowmap eingezeichnet, alles andere schwarz.
Nun wird die Glowmap mit einem Gauß’schen Blurshader in x-Richtung geblurred. Diese entstehende Textur benötigt ein drittes FBO. Um den vollständigen Gauß’schen Blur zu erhalten wird noch in y-Richtung geblurred. Dazu könnte man ein weiteres FBO verwenden. Wir entschieden uns aber dafür wieder in das FBO2 zu rendern, um Ressourcen zu sparen.
Nun müssen die Ergebnisse von FBO1 und FBO2 noch kombiniert werden.
Dabei zeichnen wir einen Fullscreen-Quad auf den Bildschirm und addieren die Texturwerte von FBO1 und FBO2 (Additive Blending).

Quellen:
- Vorlesungsfolien
- http://forum.devmaster.net/t/shader-effects-glow-and-bloom/3100
- http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html

- **GPU Particle-System (1 Punkt):**

Wo man es bemerkt:
Unser Particle-System zeigt sich dann, wenn eine Torte ein Target trifft, oder das Target zu lange am Boden liegen bleibt: die Torte zerspringt in mehrere hundert Teile.

Implementierung:

Bei der Initialisierung des Particle-Systems wird der Spawnpoint für die Partikel gesetzt und 500 Partikel werden erzeugt. Jedes dieser Partikel besitzt folgende Attribute: Typ, Position, Alter, Geschwindigkeit, Lebenszeit und Flugrichtung.
Der Typ gibt an, ob das Partikel noch lebt, oder schon tot ist. Die Position ist die Position im 3D-Raum, Alter gibt an, wie viele Millisekunden das Partikel bereits lebt, Geschwindigkeit bezieht sich auf die Fluggeschwindigkeit des Partikels, Lebenszeit sagt aus, wie lange das Partikel leben wird (Es stibt, wenn Alter > Lebenszeit), Flugrichtung spezifiziert die Richtung, in die das Partikel derzeit fliegt. Jedes Partikel erhält dabei individuelle Zufallswerte, um das ganze etwas realistischer wirken zu lassen. Die Zufallswerte stammen dabei aus der STL (mt19937).

Beim Update der Partikel wird nun jedes Partikel auf der GPU durch einen Transform-Feedback-Buffer geupdated. Das geschieht folgendermaßen: Die Partikel befinden sich auf der GPU in BufferA. Sie werden durch einen VertexShader gegeben, gelangen zu einem GeometryShader, der alle Attribute updated und schließlich in BufferB schreibt. Im Update wird das Alter des Partikels erhöht, es möglicherweise als tot markiert, die Position geupdated und die Flugrichtung so angepasst, dass der Eindruck entsteht, die Partikel würden von der Gravitation angezogen.
Nach dem Update werden die Buffers geswaped, sodass BufferA zu BufferB wird und umgekehrt. Das swaping ist notwendig, da im nächsten Frame nun wieder BufferA auf BufferB schreiben kann, damit sich die Partikel immer weiter updaten.

Nun muss also BufferA gezeichnet werden. Dies geschieht mit einem einfachen VertexShader, der die Daten nur weiterreicht, einem GeometryShader, der aus den Partikelpositionen einen Quad generiert (welches immer zur Kamera schaut, damit man das Quad immer sieht) und einem FragmentShader, welcher auf das erzeugte Quad eine Textur gibt, die so aussieht als würde die Torte in viele Einzelteile zerspringen.
Alle Partikel besitzen eine Textur mit Alpha-Werten, um keine Vierecke zu zeichnen, sondern in unserem Fall Kreise.

Quellen:
- Vorlesungsfolien
- http://ogldev.atspace.co.uk/www/tutorial28/tutorial28.html

- **Shadow-Mapping (+PCF) (1.5 Punkte):**

Wo man es bemerkt:
Unsere Shadows bemerkt man in der gesamten Szene mit Ausnahme der Hand, die keine Schatten wirft, aber auf ihr können Schatten gecastet werden, und der Torte, die von der

Hand gehalten wird. Das heißt, dass alle restlichen Objekte aus unserer Szene sowohl Schatten werfen können, als auch im Schatten liegen können.

Implementierung:
Das von uns implementierte Shadow Mapping verfügt über 2 Passes. Das heißt, dass unsere Szene aus 2 verschiedenen Perspektiven gerendert wird: ein mal aus der Perspektive der Lichtquelle und ein zweites mal aus der View/Camera Perspective.
Im ersten Pass wird die gesamte Szene aus der Sicht der "Sonne" gerendert, wobei aber nur die Depth Werte in einer Textur gespeichert werden. Diese Textur wird in einem Framebuffer gespeichert und dann dem Shader übergeben.
Im zweiten Pass wird die Szene aus der Perspektive der Camera gerendert. Beim Zeichnen der Pixel durch den Shader wird jedoch überprüft ob die Entfernung von der Lichtquelle zum entsprechenden Shadow Map Wert gleich der Entfernung von der Lichtquelle zum eigentlichen Pixel ist. Falls das der Fall ist, liegt der Pixel in der Sonne und muss nicht schattiert werden. Falls aber die Distanz zur Shadow Map kürzer als die Distanz zum Pixel selbst ist, liegt der Pixel im Schatten und muss dünkler gezeichnet werden.
Durch das PCF erhaltet man einen schöneren, nicht so abrupten Übergang vom Schatten in die beleuchteten Flächen. Dies setzt die Verwendung einer sampler2DShadow statt einer normalen Textur (sampler2D) im Shader voraus.

Quellen:
- Vorlesungsfolien zu Shadow Mapping
- http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

- **Cell-Shading (+ Contoures durch Backfaces) (1 Punkt):**

Wo man es bemerkt:
Das Cel Shading verleiht unserer gesamten Szene einen Comic Stil. Alle Objekte in unserer Szene verfügen über eine schwarze Kontur und über eine kleinere Range an Farben, mit Ausnahme der Targets, welche bloß über eine schwaze Kontur verfügen. Die Targets verwenden für die Farben den normalen Texture Shader. Wir haben uns dazu enrschloßen, weil es so besser ausgesehen hat.

Implementierung:
Für die Implementierung des Toon Effekts haben wir auch 2 Passes benötigt:
Im ersten Pass, zur Herstellung der Konturen, wurde das Objekt in einer einzigen Farbe gezeichnet, schwarz, und dabei wurde jede Fläche entlang ihrer Normale um einen bestimmten Offset verschoben. Dadurch zeichnet man eigentlich ein vergrößertes Objekt. Es ist jedoch auf 2 Sachen zu achten:

1) Im ersten Pass werden nur die Backfaces gezeichnet.
2) Damit an den Ecken keine Löcher entstehen, muss man die gemittelten Normalen verwenden.
Im zweiten Pass, zur Herstellung des eigentlichen Cel Shading, wird das Objekt in seiner ursprünglichen Größe gezeichnet, jedoch diesmal werden nur die Frontfaces gezeichnet. Außerdem werden alle Intensitätsstufen der Textur auf eine kleinere, selbst definierte Anzahl an Intensitätsstufen gemappt.

Quellen:
- http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=15

Wir haben uns für dieses Effekt entschieden, da es gut passte sowohl zu unserem Modellierungsstil der Objekte, als auch zur Spielidee.

Features

- Effekte (Bloom, GPU-Particle-System, Shadow-Mapping, Cel-Shading)
- Modelloader, der beliebige Strukturen ins Spiel lädt
- Sound-Engine, welche beliebige WAV-Files abspielt
- Physik-Engine, welche Collision-Detection und Gravitation liefert
- Hierarchische Animation
- Text-Rendering
- Phong-Lighting
- Frei bewegbare Kamera (eingeschränkt durch Gravitation)
- Config-File, mit dem sich die Auflösug ändern und ein Fullscreen-Mode aktivieren lässt
- Coole Models, welche mit einer einzigen Ausnahme von uns selbst entwickelt wurden
- 2 Levels (Ein Sommerlevel und ein Winterlevel), die sich zyklisch wiederholen, während der Schwierigkeitsgrad stetig steigt
- Einige 2D-Graphiken als Overlay über die 3D-Szene
- Alpha-Blending bei sämtlichen Overlays und dem Partikel-System
- Mip Mapping
- Ein lustiges Spiel, mit Suchtfaktor

Beleuchtungsmodell

Bei uns gibt es eine einzige Lichtquelle: Die Sonne.
Sie beleuchtet unsere 2 Levels. Dabei wird Blinn-Phong-Shading eingesetzt, um mit ambienten-, diffusen- und spekularen-Licht eine möglichst schöne Beleuchtung zu generieren. Es werden generell alle Objekte der jeweiligen Szene beleuchtet und sie werfen auch alle einen Schatten.

Tools used to create the Models

Alle Objekte aus unserer Szene wurden von uns in *Blender* modelliert, mit Ausnahme der Hand, welche wir von der Seite *turbosquid* haben. Diese mussten wir jedoch mit Blender erweitern, damit sie in unser Spiel passt.

Von uns modelliert:

- vier verschiedene Tortenarten
- die Bäume
- die Targets
- die Schneemänner
- die 2 Landschaften
- die SkyBox

Heruntergeladen, aber eigenständig erweitert:

- die Hand

Quelle:
http://www.turbosquid.com/3d-models/free-human-hands-feet-foot-3d-model/587188
(die Hand ganz rechts)

Hierarchical Animation

Wo man es bemerkt:
Die Animation kann in Pie Wars Reloaded an den drei Schneemännern aus der Winterlandschaft (beispielsweise Level 2) betrachtet werden. Die drei Schneemänner winken synchron mit beiden Händen.

Implementierung:
Die beiden Arme der Schneemänner stellen *children* des Rumpfes dar (der Rumpf stellt den *parent* dar). Diese Hierarchie haben wir in Blender erstellt (so wie das Modell selbst). In unserer Implementierung des Spiels wird das Modell auch in Form einer Baumstruktur

gespeichert. Dies ermöglichte die Selektierung der 2 children (die Arme) und deren Rotation durch eine Rotationsmatrix (glm::rotate). Die resultierende rotierte Matrix wird dann mit der Transformationsmatrix des parent multipliziert.

Experimenting with OpenGL

Wir haben in unserem Spiel an sehr vielen Stellen *Vertex Buffer Objects (VBOs)* benutzt, um Daten in der Grafikkarte zu speichern.
Dabei haben wir nicht nur auf normale VBOs beschränkt, sondern auch Interleaved-Buffers verwendet, um unsere OpenGL-Kenntnisse zu stärken.

Vertex Array Objects (VAOs) haben wir selbstverständlich auch benutzt. Der OpenGL Core 3.3 verlangt auch die Benutzung von VAOs und ohne diese wurden immer OpenGL-Fehler ausgegeben.
VAOs sind auch sehr angenehm, da sie alle VBOs, IBOs und AttribPointer schön kapseln.

Wir haben *Frame Buffer Objects (FBOs)* benutzt, um ShadowMapping und Bloom zu implementieren. Im Fall des Benutzens von FBOs wird die Szene nicht auf das Display gerendert sondern in eine Textur.

Key Mapping:

- F1 - -
- F2 - Frame Time on/off
- F3 - Wire Frame on/off
- F4 - *Texture-Sampling-Quality*: Nearest Neighbor/Bilinear
- F5 - *Mip Mapping-Quality*: Off/Nearest Neighbor/Linear
- F6 - -
- F7 - -
- F8 - -
- F9 - Transparency on/off

Gameplay Step-by-step Instructions

➔ In jedem Level muss man alle im Level vorhandenen Targets mit Torten abschießen, damit man in das nächste Level kommen kann.
➔ Dabei ist zu achten, dass man mit den Torten die man zur Verfügung hat auskommt
➔ Wenn die Torten zu Ende sind (und die Targets noch nicht), oder wenn die Zeit aus ist, hat man das Spiel verloren.

- ➔ Das Spiel verfügt über 2 unterschiedliche Landschaften:
  - ❏ eine Sommerlandschaft
  - ❏ und eine Winterlandschaft,

  welche sich immer abwechseln. Das heißt, dass der erste Level in der Sommerlandschaft stattfinden wird, der zweite Level in der Winterlandschaft, der dritte Level wieder in der Sommerlandschaft, usw.
- ➔ Dabei stehen dem Spieler jedoch immer 2 Torten weniger zur Verfügung: das heißt im ersten Level hat man 20 Torten, im zweiten Level jedoch nur mehr 18, im dritten Level hat man 16, im vierten 14, usw bis zu 10 Torten.
- ➔ Die höchste Schwierigkeitsstufe unseres Spiels stellen die Level dar, in denen man bloß verfügt über 10 Torten verfügt. Dabei muss man trotzdem 10 Targets abschießen. Das heißt, dass der Spieler keine Fehler machen darf um in das nächste Level zu kommen.
- ➔ Man kann das Spiel also nicht verlieren, man kann aber eine immer bessere High Score erreichen (die High Scores werden jedoch nicht gespeichert in der aktuellen Game Version).

Bemerkung:
Die hierarchische Animation, die dadurch dargestellt ist, dass die Schneemänner mit ihren Arme winken, kann nur im zweiten Level gesehen werden!

Controls

Unser Spiel ist folgendermaßen steuerbar:

W,A,S,D	Vorwärts, links seitwärts, zurück, rechts seitwärts bewegen
Mausbewegung	Umsehen - Rotation um die eigene Achse
Linke Maustaste - Halten	Tortenwurf aufladen (je länger man die Taste hält, desto stärker wird der Wurf)
Linke Maustaste - Loslassen	Torte wird geworfen
F2	FPS on / off
F3	Wireframe-Modus on / off
F4	Texture Sampling Quality: Nearest Neighbor / Bilinear
F5	Mip Mapping Quality: off / Nearest Neighbor / Linear

F9	Alpha-Blending on / off

Zusätzliche Libraries:

Wir haben folgende Libraries in unserem Spiel benutzt:

- Assimp: zum Laden von komplexen Objekten. Die komplexen Objekte wurden dann in einem Zwischenschritt in eine eigene hierarchische Struktur geladen, wie gefordert.
  Quelle: http://assimp.sourceforge.net/main_downloads.html
- DeviL: um Laden von Texturen
  Quelle: http://openil.sourceforge.net/download.php
- OpenAL + ALUT: zum Laden und Abspielen von Sounds im Spiel.
  Quelle: http://kcat.strangesoft.net/openal.html
- GLEW: um Laden des OpenGL-Kontextes
  Quelle: http://glew.sourceforge.net/
- GLFW: zur Erstellung des Fensters und für sämtliche Tastatureingaben
  Quelle: http://www.glfw.org/download.html
- Physx: um Physik im Spiel darzustellen
  Quelle: https://developer.nvidia.com/physx-sdk
- Freetype: zum erzeugen von Glyphen-Bitmaps
  Quelle: http://www.freetype.org/download.html
- GLM: für einige Vektor- und Matrixberechnungen
  Quelle: http://sourceforge.net/projects/ogl-math/