# Slim Jim

## Kurzbeschreibung der Implementierung

* **Gameplay (20 Punkte)**

Die Spielfigur (Jim) wird in einen Level gesetzt, wo sie sich bewegen kann. Ziel des Spiels ist es so schnell wie möglich und mit so vielen Punkten wie möglich ins Ziel zu kommen. Punkte kann man durch essen der gesunden Speisen erlangen. Durch das essen von ungesundem verliert man wieder Punkte. Zuerst beginnt man langsam und kann nur mit den Händen rudern bis man eine gewisse Punkteanzahl erreicht hat. Danach wird das Hovercraft aktiviert und man kommt viel schneller voran und kann unter anderem auch diverse Abkürzungen nehmen.

* **Effekte (20)**

Folgende Effekte wurden implementiert (laut Liste):

|  |  |  |
| --- | --- | --- |
| Effekt | Sichtbar wo? | Punkte |
| Shadow Volumes mit zfail-Technik und Self Shadowing auf den Objekten. | Überall im Level, mit F7 abschaltbar | 3 |
| Spotlights/projizierte Texturen (Eine dynamische Textur die mit projektivem Texturing auf Geometrie aufgebracht wird.) | An der Wasseroberfläche | 2 |
| Per Pixel Lighting (Phong-Beleuchtung im Fragment Shader.) | An fast allen Objekten, besonders am Spieler | 2 |
| Wasser (Sich dynamisch ändernde Wellen-Geometrie und eine Texturanimation) | Überall im Level | 1 |
| Planare Spiegelungen (Die Szene wird gespiegelt mit Stencil Buffer oder in eine Textur gerendert und dann auf eine planare Oberfläche aufgebracht. Dieser Effekt lässt sich beispielsweise für Spiegel verwenden, und lässt sich auch sehr gut mit Wasser kombinieren.) | Überall im Level | 2 |
| Vertex Skinning (Eine komplette Bonestruktur wird aus einem Model File geladen und über weighted Indices dem Model zugewiesen. Geschieht die Interpolation auf der GPU in einem Vertexshader gibt es einen Zusatzpunkt.) | Am Spieler und an jedem bewegten Objekt im Spiel. | 2 |
|  | **Summe:** | **12** |

*ShadowVolumes*

ShadowVolumes wurden so implementiert, dass die Objekte bereits vorab mit Quads zwischen den Dreiecken ausgestattet werden und diese dann auf der GPU entsprechend gestreckt werden. Die Zeichnung im StencilBuffer wird mittels Zfail Technik gemacht, da der Spieler oft in ein Schattenvolumen kommen kann. Auch die Schattierung von animierten (geskinnten) Objekten ist möglich.

Referenzen: ShaderX (Wolfgang Engel, [www.shaderx.com](http://www.shaderx.com))

*Wasser*

Beim Wasser wird die Reflection und die Refraction Textur uebereinandergeblendet. Der Anteil der Refraction/Refraction wird mittels einer Snell-Approximation gemacht:

wobei

Die Brechung wird dadurch berechnet, dass die Höhe der Wellen auf dem Bildschirm (projected) mit einem Faktor multipliziert wird und dann die Texturkoordinaten für die projizierte Textur entsprechend abgeändert werden.

Das Clipping bei der Reflexion wird mittels einem Clipplane gelöst, da die Verwendung des Zbuffers und genauem Clipping zu langsam war.

Referenzen: ShaderX²:Tips and Tricks with DirectX9 (Wolfgang Engel, [www.shaderx2.com](http://www.shaderx2.com))

* **Animierte Objekte (10)**

Man kann die Spielfigur mit der Tastatur steuern. Diese wird bei einer Vorwärtsbewegung mittels Software VertexSkinning animiert. Alle anderen Objekte im Level, die sich in irgendeiner Form bewegen werden mittels Hardware VertexSkinning animiert (Banane, Kirsche, Windmühlen).Weiters bewegt sich das Wasser (Wellen) mittels einer Geometrietransformation (Hardware).

* **Frustum Culling (5)**

Frustum Culling wurde mittels linearer Listen implementiert und kann mit der Taste F8 deaktiviert oder wieder aktiviert werden.

* **Experimentieren mit OpenGL: VBO / Display Lists (5)**

Beide Varianten wurden so implementiert, dass während der Laufzeit mit der Taste F6 umgeschalten werden kann.

* **Experimentieren mit OpenGL: Wireframe Modus / Framerate und andere Debug Ausgaben (5)**

Der Wireframe Modus kann mit der Taste F3 aktiviert werden. Dabei werden die Linien der Schatten automatisch in blau dargestellt (können mit der Taste F7 deaktiviert werden). Die Framerateausgabe kann mit der Taste F2 aktiviert oder deaktiviert werden.

* **Funktionierende Steuerung (5)**

Die Steuerung der Spielfigur erfolgt mit der bekannten W,A,S,D-Steuerung. Dabei wurden physikalische Gleichung für die Geschwindigkeit/Beschleunigung eingebaut. D.h. dass die Spielfigur nicht 100% der Geschwindigkeit in Null Zeit von einer in die andere Richtung transferieren kann. Hat die Spielfigur eine bestimmte Anzahl an gesunden Früchten gegessen, so wird das Hovercraft aktiviert und die Figur kann schneller fahren.

## Features des Spiels

Folgende Features wurden implementiert:

* Mathematik (Vektor, Matrix, Ebene, Quaternion)
* NVIDIA Cg Shader Manager
* Vertex Buffer Objects, VertexArrays
* Texturloader (eigen: BMP und TGA, jpeglib für JPG)
* Modelloader(Quake3 md5mesh/md5anim)
* VertexSkinning (Software UND Hardware)
* BoundingBoxes
* Kollisionserkennung BoundingBoxes, Strahlen
* Kollisionserkennung detailliert mit PhysX
* Wasserspiegelung
* Wasserbrechung
* Wassergeometrie (animiert, Hardware)
* Skybox
* FrameBuffer Objects
* Materialformat/Manager
* ShadowVolumes mit Zfail
* Per Pixel Lighting

## 3D-Models

Alle 3D-Modelle wurden selbst in 3ds max erstellt. Einige wurden nach Tutorials im Internet nachmodelliert. Auch das Level wurde in 3ds max modelliert und mit eigens erstellten MAXScript Plugins exportiert.

## Zusätzliche Libraries

* GLUT
* GLEW
* jpeglib
* PhysX
* md5mesh Loader von Richard Cookman
  + Könnte Meshes auch so darstellen, wurde aber nur zum Auslesen der Dateien \*.md5mesh, \*.md5anim verwendet
  + War leider fehlerhaft und musste von mir korrigiert werden (Normalen).
  + http://tfc.duke.free.fr/coding/md5-specs-en.html