CG2/3 LU Finale Abgabe
WHACKER SMACKER

Michael Nejez – 0426935 - 532
Florian Felberbauer – 0625034 - 532

- Featurelist
    - **Datenstrukturen:**

      Für die Modelle existiert eine Datenstruktur, die die Meshes für die gerenderten Models, sowie die für die Berechnung der Convex Shapes benötigten Meshes, die wir an PhysX übergeben, enthält. Zusätzlich werden bei der Instanzierung der Models einige für den weiteren Spielverlauf notwendige Daten (hauptsächlich relevant für die PhysX-Simulation) aus den .ini Files geladen, die sich im jeweiligen Folder der Models befinden.

      Auch für Texturen besteht eine Datenstruktur, die Texturen in verschiedenen Qualitätsstufen liefert, um den Anforderungen der Aufgabenbeschreibung im Sinne von Umschalten der Texturqualität sowie der MipMap-Modes gerecht zu werden.

    - **Loader:**

      Zum Laden der Models haben wir einen Loader implementiert, der XML-Files im OgreXML Format liest. Beim Laden der Models werden zusätzlich Boundingboxen und Bounding Spheres berechnet, um diese später bei Bedarf zur Verfügung zu haben. Da das Laden innerhalb weniger Sekunden vonstatten geht, haben wir von einer Serialisierung zur Beschleunigung des Ladevorgangs abgesehen.

    - **Kamera:**

      Die Kamera ist eine 3rd -Person Kamera die in der horizontalen Ebene beliebig rotiert werden kann, in vertikaler Richtung ist die Rotation eingeschränkt (hier haben wir uns an gängigen Spielen orientiert). Bewegt sich der Roboter des Spielers, so folgt ihm die Kamera. Durch das Mausrad lässt sich die Szene ein wenig zoomen.

- **Bewegte Objekte:**

  Die Roboter werden mit 3dsmax erstellt und mit OgreMAX als .xml Dateien passend für OpenGL exportiert. Über den oben genannten MeshLoader werden sie geladen und in einem MeshManager entsprechend verwaltet, um mehrfaches Laden derselben Mesh zu verhindern.

  Die Bewegung wird über PhysX kontrolliert (W,A,S,D und Mouselook zum Steuern des Spieler-Roboters).

  Der feindliche Roboter versucht selbstständig in Richtung des Spielers zu fahren um ihn zu attackieren. Ebenfalls über PhysX erfolgt eine Kollisionsüberprüfung zwischen den Robotern, sodass sie Hitpoints verlieren wenn es zu einem Kontakt kommt. Per Raycasting detektiert die KI Hindernisse wie Brücken und Fallen und versucht diesen auszuweichen. Um die Berechenbarkeit einzuschränken kommt es hier allerdings auch vor, dass sich der Gegner in einer Grube versenkt.

  Fällt ein Roboter in eine der Fallen, so werden diese geschlossen. Ist der Gegner hineingefallen, werden die Fallen wieder geöffnet, sobald ein neuer Gegner gespawned ist.

- **Animationen:**

  Drückt der Spieler die Spacebar, bzw. kommt der Gegner in Reichweite wird eine Animation abgespielt. Beim Flipper-Roboter hebt sich das Dach wie eine Klappe, beim Gegner wird ein Speer ausgefahren um den anderen Roboter zu beschädigen. Die Animation wird im Immediate Mode gerendert. Die Animation wurde mittels Keyframe-Animation implementiert. Die Keyframes werden mittels 3dsmax exportiert.

- **Texture Mapping:**

  Texturen werden mittels eines Texture Loaders aus .raw Dateien geladen, welche mit Photoshop erstellt wurden. Ebenso wie die Meshes werden die geladenen Texturen werden von einem Texture Manager verwaltet, um mehrfaches Laden zu unterbinden.

  Alle Objekte und die Skybox sind texturiert.

- **Beleuchtung/Materialien:**

  Die Beleuchtung erfolgt über eine einzige positionierte Lichtquelle die die Sonne simulieren soll. Sie befindet sich genau im hellen Fleck der Skybox, um dem Eindruck einer Sonne gerecht zu werden.

  Alle Objekte haben Materialien zugewiesen und werden beleuchtet. Durch Per Pixel Lighting wurde Phong-Shading implementiert, um den Realismus zu steigern. Dabei ist die Arena am wenigsten, die Roboter schon mehr und die Satellitenschüssel am meisten glänzend.

- **Collision Detection:**

  Die Collision Detection wird mittels PhysX realisiert. Die Roboter bestehen aus konvexen Meshes, an die WheelShapes attached sind um ein möglichst authentisches Fahrverhalten zu erreichen. Die Beschleunigung wird über motorTorque und die Lenkung über steerAngle´s erzielt. Beide Roboter verfügen über Heckantrieb und einer Lenkung an der Vorderachse. Der Flipper-Roboter besteht hierbei aus 4 WheelShapes, der Reptilienbot aus 3.

- **Font Renderer:**

  Zur Anzeige von Meldungen am Bildschirm haben wir nach dem Posting von Kustl im Informatikforum einen Font-Renderer geschrieben, wobei aus einem Textur-File die notwendigen Zeichen auf Quads gerendert werden. Das Spacing zwischen den Zeichen wird aus einem .ini – File geladen und die Positionierung wird mittels eines Enums übergeben.

- **Spezialeffekt - Lensflares:**

  Blickt man in die Sonne, werden Lens Flares gerendert.
  Hierbei wird mittels Frustum Culling überprüft ob sich die Sonne im Blickfeld befindet. Hier haben wir uns an folgenden Tutorials orientiert:
  Fast OpenGL – rendering of Lens Flares:
  http://www.opengl.org/resources/features/KilgardTechniques/LensFlare/
  Nehe Tutorial für 3D-Lensflares:
  http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=44
  Frustum Culling Tutorial:
  http://wiki.delphigl.com/index.php/Tutorial_Frustum_Culling

- **Spezialeffekt – Per Pixel Lighting:**

  Über einen Vertex- und Fragmentshader werden die ambienten, diffusen und spekulären Anteile berechnet und mit der Textur kombiniert. Durch Unstimmigkeiten zwischen OpenGL-Ergebnis und 3dsmax-Model gibt es in der Arena in Höhe der Brücken jeweils eine Kante die quer durch die Arena verläuft. Hier treten Artefakte während der Beleuchtung auf, die wir nicht beseitigen konnten (das 3dsmax File enthält diese Kanten nicht). Bei der Programmierung haben wir uns an verschiedene Tutorials gehalten:
  Lighthouse 3D: http://www.lighthouse3d.com/opengl/glsl/index.php?lights
  Delphi GL Wiki: http://wiki.delphigl.com/index.php/shader_PerPixelLighting
  Nightspawn Shader Tut: http://nightspawn.com/files/gltut/paper.pdf

- **Spezialeffekt – Partikelsysteme:** Das Öl, dass aus den Robotern spritzt wenn diese getroffen werden, wurde mittels Physx-Particle Emitter gelöst. Die Implementierung orientiert an dem diesbezüglichen Tutorial der Training Programs. Das Partikelsystem für den kleinen Wasserfall wird nur gerendert und besteht nicht aus Physx – Actors. Es benutzt die GL_POINT_SPRITE_ARB um die Funktionalität von Billboards zu bekommen und orientiert sich an den folgenden Tutorials:
  Lighthouse 3d Billboarding:
  http://www.lighthouse3d.com/opengl/billboarding/index.php3?billCheat2
  Sprites and Billboarding:
  http://goanna.cs.rmit.edu.au/~gl/teaching/Interactive3D/2009/lecture9.html
  Nehe Particle Systems Tutorial:
  http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19

- **Rendering allgemein und Funktionstasten:**

  Wie unter dem Punkt „Experimentieren mit OpenGL“ verlangt, kann bei Objekten wo dies sinnvoll ist, mittels den genannten Funktionstasten sowohl die Texturqualität als auch der MipMap Mode umgeschalten werden. Weiters können FrustumCulling, Alpha Blending, Wireframe-Modus, FPS-Anzeige und Displaylists ein/ausgeschalten und der Rendermodus zwischen Immediate Mode, VBO und Vertex Arrays umgeschalten werden. Mit F1 gelangt man überdies hinaus in den Pause-Modus, der ebenfalls mit F1 wieder verlassen wird.

- **Spielziel:**

  Das Spielziel besteht darin, den feindlichen Roboter so lange zu bashen, bis die HP auf 0 gesunken sind, der Gegner in einer Falle oder im Wasser verschwunden, oder aus der Arena rausgeflogen ist. Ist der Enemy besiegt, scheint ein 3 sekündiger Countdown auf, nachdem das Spiel mit einem neuen, stärkeren Enemy weitergeht. Pro Runde erlebt der Enemy einen Gain der HP um 10% und wird ein wenig schneller. Fällt man selbst in eine Falle oder ins Wasser, wird aus der Arena rausgekickt oder verliert solange HP bis diese auf 0 sind, so ist das Spiel aus und ein „Game Over" – Text wird angezeigt. Ebenso läuft ein Counter mit, an dessen Ende der Spieler verloren hat. Allerdings bekommt man für jeden Kill eine gewisse Bonuszeit gutgeschrieben.

- Sonstiges

  - **Zusätzliche Libraries:**

    Neben den auf der CG23 LU-Website empfohlenen Libraries Physx, GLFW, sowie GLEW und OpenAL benutzen wir nur die XML Parser Library (von Frank Vanden Berghen): http://www.applied-mathematics.net/tools/xmlParser.html

  - **Tutorials:**

    Bei der Programmierung dieser Abgabe haben wir uns natürlich an Tutorials orientiert. Die Links dazu sind die folgenden:

    Nvidia PhysX SDK - http://developer.nvidia.com/object/physx_downloads.html
    Nvidia Developer Forums – http://developer.nvidia.com
    Swiftless - http://www.swiftless.com
    Videotutorials rock - http://www.videotutorialsrock.com
    Gamedev - http://www.gamedev.net ,
    insbesondere Nehe Productions - http://nehe.gamedev.net
    OpenGL Redbook - http://glprogramming.com/red
    NightSpawn - http://www.nightspawn.de/files/gltut/html/node47.html
    Lighthouse 3D – http://www.lighthouse3d.com
    OpenGL – http://www.opengl.org
    3D-Total – http://www.3dtotal.com

- **Externer Code:**

  Wir verwenden nur die XML Parser Library, die wir nicht verändert haben.

- **Steuerung, sonstige Informationen**

  Die Steuerung wurde mittels Polling realisiert. Die Abfrage der Zusatzfunktionen wurde mit einem Keyboard-Callback implementiert. Die genaue Tastenbelegung bitte dem readme.txt entnehmen

- **Models**

  Alle Models wurden von uns mittels 3DS MAX erstellt und mittels dem Ogre Exporter in das OgreXML Format exportiert.