HAMSTER FLIGHT

- GET ROLLIN' -

Dokumentation zur 3. Abgabe

Computergrafik 2 Übung 2010

Das Spiel

Spielziel:

Aufs einfachste heruntergebrochen, geht es im großen Gesamtbild darum, einem Hamster, Aubrey, in seinem hochtechnisierten Hamsterball zur Flucht aus einen Versuchslabor zu verhelfen. (Für genauere Hintergrundstory: siehe 1. Abgabe).
Gibt es zwei einfache Levels spielbare Levels. Ziel dieser beiden ist es, vom Startpunkt aus zur einer Tür, die sich irgendwo im Level befindet, zu gelangen. Um diese Ziel zu erreichen sind mehrere Aufgaben zu erfüllen.
Ein 3. Level ist eingebaut, kann betrachtet werden, aber nicht gespielt.
(Ein 4. Level war bereits fertig modelliert, konnte aber nicht mehr eingebaut werden)

Zu beachten sind:

- Die *Maiskörner*, die im Level verteilt sind. Diese bringen Extrapunkte und ab einer gewissen gesammelten Anzahl ein wenig Lebensenergie.
- Die *Würfel*, die im Level verteilt sind:
    - Blaue Würfel: müssen meistens einfach nur verschoben werden, um an die Maiskörner darüber heranzukommen.
    - Rote Würfel: müssen irgendwie von der Platform gestoßen werden, um innerhalb des Levels etwas freizuschalten.
    - Grüne Würfel: müssen auf einen bestimmte Stelle verschoben werden, um eine Aktion auszulösen.
- Die *Gegner*, Aufziehmäuse und Spielzeugroboter, die Aubrey gefährlich werden können.

- **Level 1:**
    - Der Hamsterball ist hier nur ausgestattet mit dem Basismodus. (Die weiteren Modi müssen erst freigeschalten werden; Beschreibung siehe unten)
    - Der Weg für über die Planken auf eine Plattform. Darauf befindet sich ein roter Würfel, der hinuntergestoßen gehört. Danach erscheint die Planke zur 2. Plattform.

    - Dort muss der grüne Würfel auf seinen gekennzeichneten Platz geschoben werden.
    - Erst danach kann man durch die Türe ins nächste Level.
- **Level 2:**
    - Zuerst muss man nach rechts, auf die Plattform, an den Mäusen vorbei. Dort erhält man das Jump-Upgrade für den Hamsterball.
    - Danach muss man wieder zurück auf die Startplattform, von dort aus dann nach links weiter, über die Planken, am Roboter vorbei, auf die Plattform, wo sich ein Stiegenaufgang befindet.
    - Auf dem Stiegenaufgang liegt ein roter Würfel, der hinuntergeworfen gehört. Dann werden auf der Plattform, die über den langen Gang gerade aus von der Startplattform erreichbar ist, der blaue und ein weiterer roter Würfel freigeschalten.
    - Man muss dann zu diesem weiteren roten Würfel gelangen und ihn von der Plattform stoßen, um eine Planke freizuschalten die zur Ausgangstür führt.
- **Level 3** (wenn es spielbar gewesen wäre):
    - Nach links über die Planke, dorthin wo die Felder mit den Fragezeichen sind.
    - Man weiß nicht, auf welche Felder man draufsteigen darf und wo nicht. Zunächst muss der grüne Würfel auf eines der zugehörigen Felder geschoben werden. Danach springt man auf "JUMP HERE". Je nach dem, wo der grüne Würfel steht, wird eine Hälfte des begehbaren Weges angezeigt.
    - Ist man auf der anderen Seite erhält man das Spike (Gravity) -Upgrade für den Hamsterball. Damit muss man nun von der Startplattform aus in Standardblickrichtung nach vorn. Dort gibt es dann "Vertical Bowling", d.h. man muss sich hinunterstürzen, auf in den Spike-Modus schalten und so die Würfelpyramide weiter unten zerstören und mind. 6 Würfel davon von der Plattform stoßen.
    - Schafft man es nicht, werden die Würfel zurückgesetzt und man muss zurück zur Startplattform, und es noch einmal versuchen.
    - Hat man die "Vertical Bowling"-Challenge geschafft, muss man von der Startplattform aus nach hinten. Dort wurde (durch das "Vert. Bowl.") eine Planke freigeschalten, über die man jetzt weiterkann.
    - Dort erwartet dann einen ein Abhang, den man sich hinunterstürzen muss. Um den dort wartenden Robotern zu entgehen, und die Körner einzusammeln, muss man geschickt rechtzeitig den Spike-Modus an- und ausschalten.
    - Ist man unten angelangt, muss man über einen schmalen Weg bis zur Tür und versuchen, die Mäuse zu besiegen, in dem man über sie springt, dann in den Spike-Modus schaltet und sie zerquetscht.

Ballmodi:

Der Hamsterball hat verschiedene Modi, zwischen denen man hin- und herschalten kann. Aktuell gibt es vier Modi:

- *Basismodus (base mode):* Grundeinstellung des Balles, keine Sprungfähigkeit, normale Schwerkrafteigenschaften, gemäßigte Rollgeschwindigkeit.
- *Stachelmodus (spike mode):* Erhöhte Schwerkraft, lässt sich nicht normal bewegen. Eine Idee dahinter: später werden kleinere Gegner nur dadurch ausschaltbar sein, dass man im Stachelmodus auf sie herunterstürzt, im Basismodus würde man selbst verletzt werden.
- *Geschwindigkeitsmodus (speed mode):* Erhöhte Geschwindigkeit, Einsatz z.B. in Speedlevel, wo Loopings durchquert werden müssen.
- *Sprungmodus (jump mode):* Fähigkeit zu springen.

Game/Spectator mode:

In den Resources befindet sich die Konfigurationsdatei "settings.cfg". Dort gibt es die Variable "game type", die sich entweder auf "GAME" oder "SETTING" setzen lässt. Je nach dem ist das Level entweder spielbar oder man hat die Möglichkeit sich frei mit der Kamera durch die Szene zu bewegen. (Weiteres siehe Steuerung od. Abschnitt Kamera weiter unten)
Das Umschalten soll später auch direkt im Spiel möglich sein.

Steuerung:

- **Game mode:**

  Bewegung: (ausgehend von der Kamerablickrichtung)

W, Pfeiltaste nach oben	nach vorne
A, Pfeiltaste nach links	nach links
S, Pfeiltaste nach unten	nach hinten
D, Pfeiltaste nach rechts	nach rechts
Leertaste	Springen (im Sprungmodus)

  Ballmodi:

Y	Basismodus (weiß)
X	Stachelmodus (grün)
C	Geschwindigkeitsmodus (blau)
V	Sprungmodus (rot)

  Kamera: (Kamerabewegung bezieht sich auf die Postion der Kamera, nicht auf die Blickrichtung)

Mausbewegung nach rechts	Kamerabewegung nach rechts (auf Zylinder um den Hamsterball herum)
Mausbewegung nach links	Kamerabewegung nach links (auf Zylinder um den Hamsterball herum)
Mausbewegung nach oben	Kamerabewegung nach unten
Mausbewegung nach unten	Kamerabewegung nach oben
Linke Maustaste	Kamera zurück in Ausgangsposition vom Hamsterball aus

  Sonstiges:

P	Pausieren
Esc	Beenden des Spiels

- **Spectator mode (kann nur über config aktiviert werden, nicht im Spiel):**

  Kamera:

Mausbewegung nach rechts	Kamerarotation nach rechts
Mausbewegung nach links	Kamerarotation nach links
Mausbewegung nach oben	Kamerarotation nach oben
Mausbewegung nach unten	Kamerarotation nach unten
W	Kamerabewegung nach vorne auf Blickrichtung
A	Kamerabewegung nach links
S	Kamerabewegung nach hinten entgegen der Blickrichtung
D	Kamerabewegung nach rechts
Q	Kamerabewegung nach oben
E	Kamerabewegung nach unten

  Sonstiges:

F2	Aktivieren des Mauscursors
F3	Deaktivieren des Mauscursors
Esc	Beenden des Spiels

- **CHEAT KEYS**
  - Drücken von F2 oder F3 während des Startscreens ("Hit Enter to start game) führt direkt in Level 2 bzw. Level 3
  - Drücken von F1 während eines Levels führt direkt in das nächste Level
  - Drücken von Enter, wenn man gestorben ist, setzt das Level zurück

Die Implementierung

(Verweise auf den Code werden kursiv unter Anführungszeichen geschrieben, des betreffende package klein, die Klasse mit großem Anfangsbuchstaben.)

Übersicht Funktionstasten:

(In Klammer der Standard)

- F2: Framerate anzeigen an/aus (aus)
- F3: Wireframemodus an/aus (aus)
- F4: Texturequalität Nearest Neighbor/bilinear (bilinear)

- F5: MipMaps-Verwendung aus/Nearest Neighbor/biliner (Nearest Neighbor)
- F6: Verwendung von VBO an/aus (an) (KEINE AUSWIRKUNG)
- F7: Physik-Engine Debuganzeige an/aus (aus)
- F8: Frustum Culling an/aus (aus)
- F9: Transparenz an/aus (an)
- F10: Mauscursor aktivieren/deaktivieren (aktiviert)
- F11: Animation an/aus (an)

Momentane Featureübersicht:

- Diverse Datenstrukturen
- Ressourcenallokation
- Lichtquellen, Beleuchtung & Material
- Steuerung
- Kamera
- Bewegte Objekte
- Charaktermodelle
- Texturierung
- OpenGL Buffer-Objects: VBO & VAO
- Physik
- Sound
- Cel-Shading + verstärkte Umrisse
- Wireframemodus
- Font-Rendering
- Transparenz
- Frustum Culling
- Animation

Datenstrukturen:

Es werden unterschiedlichen Datenstrukturen (*"scene"*) mit variierender Komplexität zur Speicherung der Szene, Models etc. verwendet. Im Folgenden eine Auflistung:

- Szenegraph (*"Scene"*):
    - enthält Objekte, welche angezeigt werden, sowie Beleuchtungselemente
    - Der Szenegraph besteht aus Blättern (*"Spatial"*) und inneren Knoten (*"Node"*) und erlaubt hierarchische Transformationen.
    - Transformationen (lokal & global) (*"Transformation"*): Rotation, Skalierung, Translation
    - auch Verwaltung über Bullet-CollisionObjects
- Grafikmodel (*"Model"*): Wrapperklasse für folgende Eigenschaften eines Objektes:
    - Material: als Uniforms
    - Vertices: Bildpunkte, Normalvektoren & Texturkoordinaten
    - Abmessungen um die BulletCollision Shapes zu berechnen
    - Texturdaten
- Lichter (*"Light"*):
    - können in den Szenegraphen eingehängt werden
    - Der Szenegraph kann beliebig viele Lichtquellen enthalten, für jedes Modell werden die Lichter ausgewählt, die am meisten zu seiner Farbe beitragen.
- Kamera (*"Camera"*):
    - Repräsentation der Kamera

    - Es besteht die Möglichkeit, diese in den Szenegraph einzuhängen, um Physikeigenschaften in Bezug auf andere Objekte zu nutzen.

Ressourcenallokation:

Das Laden der verschiedenen Dateitypen wird von einem Ressourceloader (*"resource"*, *"Uberloader"*) übernommen.
Dieser kann folgende Typen laden:

- Konfigurationsdateien (*"ConfigLoader"*): Sammlung der generellen Konfigurationsdaten der Spielbestandteile (Models für Level und Charaktere)
- MD2-Modelle (*"ModelLoader"*): Laden der MD2-Modelle und Aufbereitung für OpenGL
- Shader (*"ShaderLoader"*): Laden und Erzeugung der Shader
- Textur (*"TextureLoader"*): Importieren von TARGA-Dateien (*.tga) mit Hilfe von DevIL und erzeugen von Texturen daraus.
- Sound (*"SoundLoader"*): Laden von Ogg Vorbis-Dateien (*.ogg) mit Hilfe von OpenAL & Ogg Vorbis-eigenen Bibliotheken.

Ressourcen:

Folgende Resourcen (*"resources"*) werden für das Spiel benötigt:

- Konfigurationsdateien (*.cfg): Können enthalten:
    - Die wichtigsten Daten für die Objekte einer Szene: verwendetes Model, Shader und Texture, Physikeigenschaften, Materialeigenschaften sowie zum Platzieren in der Szene notwendige Transformationen.
    - Eigenschaften für das gesamte Spiel oder einzelener Levels
    - Eigenschaften der Lichter: Farbe, Abschwächung, Transformation
    - etc.
    - Zum Laden der Konfigurationsdateien existiert ein eigener Parser der im Package *"config"* zu finden ist.
- MD2-Modelle (*.md2): Sämtliche Modelle wurden von uns selbst mit Blender erzeugt.
- Texturen (*.tga)
- Sounds (*.ogg)
- Shader (*.vert, *.frag): Momentan im Einsatz: simple.vert & simple.frag

Lichtquellen, Beleuchtung & Material:

Wir verwenden Punktlichter mit ambient, diffuse und specular Komponenten und konstanter, linearer und quadratischer Attenuation (Abschwächung).
Besonderheit bei den Lichtern: Man hat eine Virtualisierung der Lichter, d.h. man kann beliebig viele Lichter hinzufügen, diese werden für jedes Modell nach Stärke der Einstrahlung sortiert, und nur die stärksten Lichter werden an den Shader übergeben.
In der Abgabe werden zwei Lichtquellen verwendet.

Die benötigten Daten (bzgl. Beleuchtung und Material), die zum Zeichnen (*"Renderer", "render"*) verwendet werden und die aus den Konfigurationsdateien ausgelesen werden, werden in *"Shader"*, *"Material"* und *"Light"* verwaltet.

Bezüglich der Beleuchtung von Modellen mit harten Kanten:
Die MD2-Modelle wurden so aus Blender exportiert, dass eine per-faceNormals Beleuchtung möglich ist.

Sollte auf einer gerade Fläche, auf der keine Lichtänderung zu erwarten wäre, dennoch eine solche auftreten, so ist das ein Effekt des Zusammenwirkens von Cel-Shading und der Lichtabschwächung über die Entfernung.

Steuerung:

Die direkten Steuerungsabfragen sind in *"control"* als *"Control"* und *"CameraHandler"* gekapselt.
"Control" dient der Kapselung der allgemeinen OpenGL-Steuerungsabfragen, z.B. Tastendruck und dergleichen.
"CameraHandler" ist rein für Neuberechnung der Kameraposition ausgehend von der Mausbewegung relativ zu einem Zielobjekt verantwortlich.
"ControlLogic" (ebenfalls *"control"*) stellt die Verbindung zwischen den Steuerungsabfragen und der eigentlichen Gamelogic dar.

Kamera:

Die Kamera als programmiertechnisches Objekt befindet sich in *"Camera"* (*"scene"*).
Als Objekt in der Szene betrachtet, bewegt auf einem virtuellen Zylinder um den Hamsterball. Die Kamera lässt sich somit auf 360° um den Hamsterball herumbewegen.
Wenn in *"main"* verlangt wird, dass die Kamera aktualisiert werden soll, wird die Transformation der Kamera zum Ziel (in unserem Fall der Hamsterball) neu berechnet, und auch mit dem Ball mitbewegt. Diese Berechnung finden sich, wie bereits erwähnt in *"CameraHandler"*. Nach der Positionsneuberechnung wird mit der Methode lookAt (*"Spatial"*, da die Kamera selbst ein Onjekt im Raum ist) die Blickrichtung der Kamera auf das Ziel ausgerichtet.

Bewegte Objekte:

Wichtigstes bewegbares Objekt ist der Protagonist des Spiels, der manuell mit der Tastatur gesteuert werden kann (siehe Bedienung).
Indirekt lassen sich auch die Würfel (mit Warnschild-Textur), die auf den verschiedenen Plattformen liegen, durch den Hamsterball bewegen.

Charaktermodelle:

Jedes dynamische Objekt in unserem Spiel, d.h. der Protagonist und seine Kontrahenten, verfügen über zusätzliche Attribute, wie zum Beispiel der Gesundheit. Diese Attribute werden in den Charakterklassen (*"character"*) verwaltet.
Das Charaktermodell ist für den Hamster ist derzeit in *"Hamster"* ausprogrammiert.
Es finden sich auch weitere Implementierung für den Spielzeugroboter, die Aufziehmaus etc.

Texturierung:

- Zur Texturierung werden zunächst TARGA-Files (*.tga) im *"TexturLoader"* (*"resources"*) unter Zuhilfenahme von DevIL geladen, und als Texturobjekte verwaltet (*"Texture"*; *"texture"*), welche die OpenGL-Befehle zum Binden und Lösen der Textur kapseln und auch Befehle zur Änderung der Textur- oder MipMap-Qualität zur Verfügung stellt.

- Zu jedem Model wird dann ein solches Texturobjekt gespeichert, mit dessen Hilfe im beim Aufruf der Zeichenfunktion des Modells, die einzelnen Texturen gebunden werden können.
- Welche Objekte wie texturiert sind:
    - Protagonist (nur der Ball): Hamsterballtextur (hamsterballTex.tga)
    - Statische Levelelemente:
        - Plattformen: Fliesenboden (floorTex.tga)
        - Wände: Ziegelsteinmauer (brickwork.tga)
        - Tür: doorTex.tga
        - Jump-Item in Level 2: jumpItemTex.tga
        - Stiegenaufgang: stairsTex.tga
    - Gegner: toyRobotTex.tga (Spielzeugroboter) & wuMouseTex.tga (Aufziehmaus)
    - Maiskörner: cornTex.tga
    - Unschuldige Würfel, die von der Plattform gekickt werden: Warnschild auf verschiedenfarbigen Hintergründe (blueCube.tga, redCube.tga, greenCube.tga)

OpenGL Buffer-Objects (VBO & VAO):

Zur Verwaltung und Abarbeitung der Vertices bzw. der Vertexdaten werden bereits Vertex Buffer Objects (*"VertexBuffer"*, *"render"*) und Vertex Array Objects (*"VertexArray"*, *"render"*) verwendet.
Diese werden für ein Objekt beim Laden des zugehörigen Models erzeugt (siehe *"UberLoader"*).

Physik:

Zur Simulation von Physik wird die Bullet Physics Library verwendet und damit ein Physiksystem (*"PhysicsSystem", "physics"*) erstellt, dass es möglich macht

- allgemeine Gravitation in der Szene zu simulieren,
- Physikeigenschaften auf die Objekte der Szene wirken zu lassen, je nach dem ob sie als "dynamic objects", "static objects" oder "kinematic objects" definiert sind,
- Zusammenstöße zwischen Objekten festzustellen (Collision Detection).

Voraussetzung dafür, dass Physik Auswirkungen auf ein Objekt hat (also für die angeführten Punkte), ist, dass um das Objekt eine Hülle definiert sein muss, die quasi den virtuellen physikalischen Körper des Objekts repräsentiert.
Weiters befinden sich im Package *"physics"*:

- *"PhysicsNode"*, was einen von der Physik beeinflussten Knoten im Szenenbaum repräsentieren soll
- *"PhysicsDebugDraw"*, was für die Debuganzeige (Collision Shapes der Objekte) verantwortlich ist.

Sound:

- Es ist möglich *.ogg-Files zu laden und abzuspielen.
- Das direkte Laden findet in *"SoundLoader"* (Zugriff über *"UberLoader"*). Geladen werden die Dateien unter Zuhilfenahme von Ogg Vorbis-eignen Bibliotheken, da komprimierte Formate nicht von OpenAL allein gelesen werden können.

- Geladene Sound-Dateien werden dann in *"Sound"* (*"sound"*) verwaltet. Diese Klasse kapselt die wichtigsten OpenAL-Befehle zum Abspielen (Play, Pause, Stop) und Einstellen eines Sounds (Lautstärke, Tonhöhe).
- Derzeit verwendet werden BGMcalmViolence.ogg als Hintergrundmusik und SFXroll.ogg als Soundeffekt für den rollenden Hamsterball.

Cel-Shading (Toon-Shading) & verstärkte Umrisse

- Das Cel-Shading findet im Fragment-Shader simple.frag (resources/shaders) statt. Es handelt sich dabei genau gesagt nur um eine Diskretisierung des diffusen Lichts mit Hilfe eine 1D-Textur (aus verschiedenen Grauwerten).
- Die Daten für diese Textur werden aus der Datei celShaderTex (resources/shaders) ausgelesen, und, sobald der Shader "simple" geladen (siehe *"Shaderloader"*) wird, wird aus diesen Daten über OpenGL eine 1D-Textur erzeugt.
- Für das Zeichen der verstärkten Umrisse wird nach dem Zeichnen der Szene ein weiterer Rendervorgang durchgeführt, der nur die Umrisse der Backfaces der Objekte zeichnet (siehe *"Renderer"* -> drawOutline()).
- Der Vorgang bedient sich der shader outline.vert & outline.frag, welche am Ende einfach nur schwarze Umrisse der geforderten Dreiecke ausgeben.

Wireframemodus:

- Das Zeichnen des Wireframes befindet sich in *"Renderer"* -> draw().
- Sollte der Wireframemodus eingestellt sein, werden einfach alle Dreieck gezeichnet, ohne sie auszufüllen (also nur ihre Umrisse).

Font-Rendering:

- Font-Rendering (im Weiteren auch das Zeichnen von Bildern über der Szene) findet in *"Renderer"* -> drawImage() -> drawString() statt.
- Dafür werden einfach 2D-Texturen (die zum einen einen Font oder ein Bild erhalten) geladen.
- Diese 2D-Texturen werden dann auf einem den gesamten Bildschirm abdeckenden Quadrat gezeichnet.

Transparenz:

- Transparenz wird ebenfalls über den Renderer ermöglicht.
- Die Ressourcedateien für die Objekte in der Szene enthalten ein Flag, das angibt, ob sie transparent gezeichnet werden sollen oder nicht.
- Je nachdem wird vor dem Zeichen Transparenz in *"Renderer"* ein- oder ausgeschalten.
- Kapselung von OpenGL-Befehle glEnable(GL_BLEND) in ->enableTransparent() bzw. ->disableTransparent().

Frustum Culling:

- Frustrum Culling wurde implementiert, die Berechnungen sind zu finden in *"Camera"* -> recreateViewFrustum().

- Ist das Culling aktiviert, wird zunächst getestet (Aufruf in *"Model"* -> render()), ob sich die Bounding Sphere des Models überhaupt in der Bounding Sphere des View Frustums befindet. Diese Bounding Sphere ergibt sich aus der minimalen Kugel, die das gesamte View Frustum einschließt. Bei diesem Test findet also ein Vor-Aussortierung statt.
- Sollte die Bounding Sphere des Models innerhalb der Bounding Sphere des View Frustums liegen, wird noch ein 2. Test durchbeführt, wo bei diesmal der Test gemacht wird, ob die Bounding Sphere des Models innerhalb oder außerhalb der 6 Ebenen, die das View Frustum bilden, liegt.

Animation:

- Es ist Animation von Modelle über Keyframes implementiert, zu finden in *"Model"* (*"scene"*)
- Sowohl der Spielzeugroboter als auch die Aufziehmaus sind zu einfachen Animationen fähig.
- Allerdings senkt die Animation aus unerfindlichen Gründen extrem die Framerate.
- Daher wurde die Animation generell deaktiviert, aber als Proof of Concept finden sich im 2. Level ein Spielzeugroboter und eine Aufziehmaus, die animiert sind.

Zusätzliche Libraries:

- Bullet (Bullet Physics Library):
    - http://bulletphysics.org
    - Verwendet für:
        - Collsion Detection
        - Ingame Physics (Gravitation)
- boost (boost C++ Libraries):
    - http://www.boost.org
    - Verwendet für:
        - shared Pointer
        - Fileaccess
- GLEW (OpenGL Extension Wrangler Library):
    - http://glew.sourceforge.net/
    - Verwendet für:
        - Fenstererzeugung
        - Benutzereingaben
- glm (OpenGL Mathematics):
    - http://glm.g-truc.net/
    - Verwendet für:
        - diverse mathematische Berechnungen
- DevIL (Developer's Image Library)
    - http://openil.sourceforge.net/
    - Verwendet für:
        - Laden von Bilddateien zur Weiterverwendung als Texturen
- OpenAL (Open Audio Library)
    - http://connect.creativelabs.com/openal/default.aspx
    - Verwendet für:
        - Abspielen und Verwaltung von Sound-Dateien
- Ogg Vorbis Libraries
    - http://www.xiph.org/downloads/

- Verwendet für:
  - Laden und dekomprimieren von *.ogg Files zur Weiterverwendung mit OpenAL