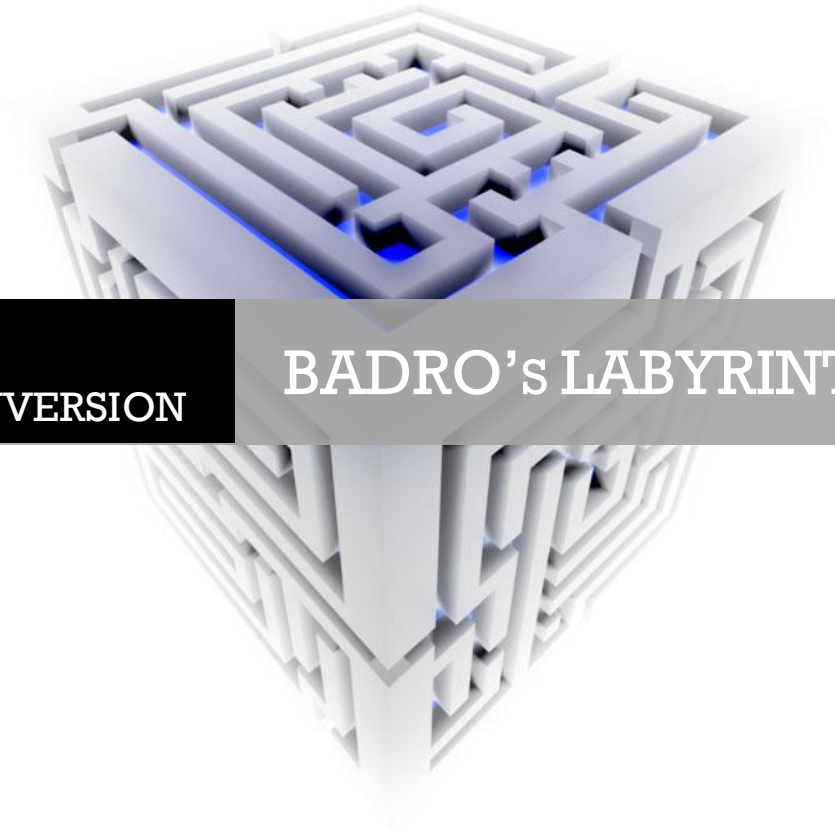
[Wählen Sie das Datum aus]



TU WIEN - STUDENTENVERSION

BADRO's LABYRINTH

3D - Labyrinth Game | Jakob Karaszek & Peter Mihalik

BADRO'S LABYRINTH

Seit der zweiten Abgabe haben wir uns sehr bemüht das Spiel in mehreren Richtungen zu verbessern. Einerseits war es unser Ziel aus einem Programm ein richtiges Spiel zu machen anderseits auch die grafischen Anforderungen der Übung zu erfüllen. Weiter wollten wir alle Bugs auf die wir gestoßen sind eliminieren.

Features des Spieles

Steuerung

Wir haben die Steuerung seit der ersten Version wesentlich umgebaut. Wir wollten die ehemalige Keyboard-Steuerung durch die Mouse-Steuerung ersetzen jedoch haben wir diese für nicht sehr angenehm und ungenau gefunden. Aus diesen Gründen sind wir letztendlich bei der Keyboard-Steuerung geblieben wobei wir dem Spieler die Möglichkeit überlassen haben die Steuerung zur Mouse nach Bedarf zu wechseln.

Die Keyboard-Steuerung haben wir „indirekter" gemacht in dem wir Beschleunigung der Bombe implementiert haben. Es dauert ein paar Sekunden bis die Bombe ihre Höchstgeschwindigkeit erreicht hat und wieder ein paar Momente bis die Bombe nach dem loslassen einer Richtungstaste zum Stilstand kommt.

Die Bewegung der Bombe auf der Fläche des Würfels passiert auf folgender Weise. Die Bombe bewegt sich immer auf der XZ-Ebene. In dem Moment, in dem die Bombe die Kante erreicht wird eine kurze CutScene abgespielt (Rein-/Raus-zoomen). In der Mitte der CutScene wir der Würfel um (ein vielfaches) von 90° um die X und/oder um die Z Achse rotiert. Der Spieler bekommt es aber nicht mit da diese Drehung zwischen zwei Frames passiert. Es erscheint ihm als würde er mit der Bombe um die Kante auf eine andere Würfelseite rollen. So ein System war sehr umständlich zu implementieren und hat uns mehrere Tage gekostet. Man muss bedenken, dass es hier 24 unterschiedliche Fälle (6 Seiten mit jeweils 4 Kanten) zu unterscheiden gibt, wobei man bei jedem dieser Fälle den Würfel richtig rotieren muss, die Bombe an die richtige Stelle verschieben muss und die Kamera richtig umdrehen muss.

Für die richtige Bewegung im Labyrinth war es notwendig ein CollisionDetection-Mechanismus zu benutzen. Für diese Zwecke haben wir die ColDet API benutzt. Wegen der Form des Labyrinthes ist es sehr umständlich eine BoundingBox zu machen, da sie die Wege in dem Labyrinth schließen würde. Es war jedoch kein Performance Problem in jedem Frame die Ballposition mit der Position vom jeden Tringle des Labyrinthwürfels (44512 Triangles!) abzufragen. Aus diesem Grund können wir ColDet nur weiterempfehlen.

Außer für Kollision der Bombe mit dem Labyrinth benutzen wir ColDet auch zur Kollisionsberechnung der Bombe mit Gegenständen, die einzusammeln sind (Dynamites). Bei den Dynamiten berechnen wir beim Einlesen des Modells ihre BoundingBox in dem wir die kleinsten und die größten X, Y und Z Koordinaten des Modells speichern und daraus eine BoundingBox aus Triangles berechnen.

Da die Bombe eine kugelförmige Gestalt hat, benutzen wir in unserem Spiel nur SphereCollisions.

Es wird vorm Rendern jedes Frames zuerst die zukünftige Position der Bombe berechnet und diese dann auf Kollisionen überprüft. Unsere erste Lösung war es im Fall einer Kollision die Bombe einfach stehen zu lassen – also nicht weiter bewegen. Dies war aber sehr lästig für den Spieler, da man die Wände des Labyrinths in den schmalen Gängen alle paar Sekunden berührt. Dieses Problem haben wir dann so gelöst, dass wir immer herausfinden ob die Wand zu der X-Achse oder der Z-Achse parallel ist und bewegen nachher den Ball nur entlang dieser Wand (in dem wir die andere Bewegungskomponente 0 setzen).

Effekte

Wir haben in unserem Spiel zwei Effekte umgesetzt:

- Particle System
- Water

Particle System

Das Particle System haben wir uns selber überlegt und keine Referenz zur Hilfe genommen. Es besteht aus zwei Klassen. Die erste Klasse heißt *ParticleSystem* und ist ein Generator von Objekten der anderen Klasse *Particle*. Mit Hilfe verschiedener Funktionen kann man mehrere Faktoren des Systems so wie seiner Partikel beeinflussen:

- Lebenszeit und ihre Varianz
- Position des Generators / der Quelle
- Die Richtung in die, die Partikel „raus geschossen“ werden und die Varianz dieser
- Die maximale Anzahl der Partikel
- Die Geschwindigkeit der Partikel
- Den Typ der Partikel (zur Zeit Cubes und Spheres)
- Die Produktion von neuen Partikeln ein bzw. ausschalten ohne das Partikel System selbst „zerstören“ zu müssen.
- Die Größe der Partikel
- Die Farben – die Startfarbe die das Partikel bei der „Geburt“ bekommt und die Endfarbe die es kurz vorm Zerstören annimmt. Zwischen diesen Werten wird zur Lebenszeit jedes Partikels interpoliert.

Neben den Farben stellen wir auch die Alphawerte der einzelnen Partikel um, also werden sie in unserem Spiel je älter desto durchsichtiger.
Dieses Partikel System wird an drei Stellen in unserem Spiel benutzt – Rauch der Bombe, Partikel die aus dem Loch rausfliegen und für die Explosion des Würfels.

Wasser

Für den Wassereffekt haben wir ein Tutorial aus dem Internet genommen und dessen Teile in der Klasse *Water* benutzt. Dieser Effekt ist eine simple sinusartige Verschiebung von Vertices entlang der Y-Achse. Es gibt drei Parameter mit den man das Verhalten der Wellen steuern kann – Frequenz, Amplitude und Phase.

Loader

Unser Spiel ist im Stande Milkshape3D Modelle einzulesen. Für diese Zwecke haben wir einen Loader aus einem Tutorial erweitert. Aus Performancegründen haben wir den Loader so umgebaut, dass er imstande ist DisplayLists zu generieren und zu benutzen, BoundingBox für ein Model zu berechnen und mit ColDet zusammenzuarbeiten. Der Loader liest die Geometrie, sowie die dazugehörige Textur (hier haben wir noch MipMapping implementiert) und die Materialattribute ein.

Fog

Aus ästhetischen Gründen haben wir in unserem Spiel statt einer SkyBox Fog benutzt. Der Startpoint und der Endpoint des Fogs werden im Laufe des Spieles dynamisch verändert, abhängig von der Entfernung der Kamera von der Bombe. Die Farbe des Fogs verändert sich bei der Endanimation (CutScene) so wie wenn „p“ gedrückt wird.

Kamera

Im ganzen Spiel wird die Third-Person-Camera benutzt. Sie folgt der Bombe und rotiert um sie beim Richtungswechsel. Die Entfernung der Kamera von der Bombe ist veränderbar (A und Z Tasten).

Modelle

In unserem Spiel haben wir insgesamt vier Modelle – die Bombe, das Labyrinth, die Dynamites und den Pfeil. Das Labyrinth ist dabei das komplizierteste Modell von allen und hat 44512 Triangles. Alle Modelle wurden mit Hilfe von Maya modelliert und als *.obj* exportiert. Diese Dateien wurden dann in Milkshape3D importiert und als *.ms3d* gespeichert. Der Würfel wurde mittels UV Mapping texturiert. Für die Rasterbilder haben wir Adobe Photoshop verwendet. Die restlichen Modelle sind nicht texturiert da wir glauben, dass sie mit diffuse color besser aussehen als mit Texturen.

Keys zur Steuerung

Das ganze Spiel wird primär mit Hilfe der Tastatur bedient. Am Anfang wällt man in der Konsole ein erwünschtes VideoMode aus. Im Spiel bewegt man die Bombe und die Kamera mit Hilfe der Arrowkeys nach vorne und man rotiert (um) die Bombe. Die F-Keys haben verschiedene Funktionen:

- F1 : Help Window
- F2 : Framerate anzeigen / ausblenden
- F3 : Wireframe- / Smoothshaded- Mode
- F4 : CCW oder CW Triangle Winding
- F5 : Front- / Back- faceculling (gleiche Auswirkung wie F4)
- F6 : Anzeigen / Ausblenden von Hintarrows (zeigen auf die Dynamites)
- F7 : Change Camera Distance (near / far)
- F8 : Keyboard or MouseControl
- F9 : Nearest / BiLinear / TriLinear (MipMapped / 4 Arten) filter
- F10: Das Benutzen von DisplayList ein- / aus- schalten

Mit *Esc* verlässt man das Spiel und mit Hilfe von A und Z zoomt man mit der Kamera in kleinen Schritten zu / von der Bombe. P heißt Pause.

Ziel des Spiels

Das Ziel des Spiels ist es mit der Bombe durch das Labyrinth durchzurollen, alle Dynamite (9 Stück) einzusammeln und die Bombe in das Loch auf der blauen Seite des Würfels zu steuern. Der einzige Gegner hierbei ist die Zeit. Die Bestzeit wird gespeichert und am Anfang des Spiels eingelesen.

Benutzte Libraries

- GLUT (http://www.xmission.com/~nate/glut.html)
- ColDet (http://sourceforge.net/projects/coldet)

Modellierungssoftware

- Autodesk Maya
- Milkshape 3D
- Adobe Photoshop CS2 zum Texturieren

Hilfreiche Quellen

- Lighthouse3D GLUT Tutorials - http://www.lighthouse3d.com/opengl/glut/
- NeHe Tutorials - http://nehe.gamedev.net/lesson.asp?index=01
- APRON tutorials - http://www.morrowland.com/apron/tut_gl.php
- C++ erfrischen / lernen für Java Leute

  http://www.cs.brown.edu/courses/cs123/javatoc.shtml

Tipps für Tester

Das Lösen des Labyrinths kann durchaus auch viel Zeit in Anspruch nehmen (zumindest ein paar Minuten) deswegen haben wir einen Cheat eingebaut. Wenn man *‚c'* drückt dann werden automatisch alle Dynamite eingesammelt und man muss nur noch die Bombe in das Loch mit dem bewegten Wasser führen und dann kommt die Endanimation.

Also um schnell durchzukommen muss man aus der Anfangsposition auf die rote Fläche die über Kante vorne (aus der Anfangssicht des Spielers) rollen. Aus dieser Sicht muss man dann über die rechte Kante auf die blaue Fläche kommen. Hier befindet sich das Loch in der Mitte der Fläche.

HUD

Es werden mehrere Infos angezeigt im Spiel – FPS, Anzahl der eingesammelten Objekte, Bestzeit und die eigene Zeit.

Schlusswort

Die Lehrveranstaltung zählt bei uns auf jeden Fall zu einer der wertvollsten und interessantesten die wir in der Studienzeit erlebt haben. Endlich mal wieder eine Lehrveranstaltung die praxisbezogen ist. Aufwendig aber trotzdem sehr interessant und schöpferisch. Danke