Apache mod_helicopter

Project report for TSBK07 Computer Graphics

Fredrik Pettersson
frepe593

Daniel Torstensson
danto629

2011-05-17, Linköping

Introduction

In our project, we wanted do create a helicopter ”flight simulator” where you control a helicopter seen in third-person perspective. The camera should be placed in a fix position, behind and slightly above the helicopter, to center the helicopter on the screen. The helicopter should be situated in a world with a rather large landscape to fly over.

Before we started, we created a specification where we specified some things we should do (mandatory) and some things that we might do (optional) if there were time.

Will do:

- Terrain to fly over, described by a heightmap image.
- Textured ground.
- SkyBox with appropriate texture (clouds).
- Collision detection with ground.
- Use of imported OBJ file for the helicopter with rotating blades.
- Third-person view (behind and above the helicopter).
- Sun light source (may the sun shine from above).

Might do:

- Endless terrain (repeated)
- Don't draw stuff that isn't in the view or too far away.
- Random objects on the ground (eg. trees, rocks).
- Sound effects (kaboom on crash and static hover sounds).
- Score objects to pick up (to improve gameplay)
- Obstacles popping up that the player needs to avoid.
- Weapons and targets to shoot.
- Selection of different camera placements, like third-person view, top view or zoom feature

Implementation

We based our work on things we created and learned during the labs we made previously in this course. That is, we continued using C and OpenGL as setup in the lab environment. Therefore, the overall structure of the programs is like in the labs. We have a main program, initializing everything and creating a (fullscreen) window and handling all display functions. Basically it’s structured in a way with different functions to render different things (renderTerrain, renderHelicopter etc). All camera control is implemented in a separate file. We also use some separate helper libraries to load textures, models and so on.

Terrain

We started off by creating the terrain. We used a heightmap image, with height data encoded as color variations, to create the variances in terrainheight. Each pixel in the heightmap represents one node in the groundplane. Between four nodes we drew two triangular polygons so they formed a square. Where the height of each corner were decided by the image data. The size of the polygons was scaled to fit a desired size of the resulting ground terrain. This resulted in a terrain rendered with lots of height variances.

We also applied a texture to the ground terrain, to make it look more like real ground. We used the same texture here as used in the skybox bottom plane, as will be described later on. This gave a nice effect of a lager world than we actually had since the mountains continued on the sides of the skybox.

Where the ground ended up flat, we applied a different texture. We used a water texture to make the flat areas to look like smaller lakes.

Skybox

We created a skybox around the camera, by simply drawing quad polygons around the center until they enclosed the camera in a cube. We found appropriate skybox textures with 6 different, but matching, textures to apply to all the sides of the cube. With every camera movement, we also moved the skybox as much, to always have the camera centered in the skybox so that the helicopter never gets outside “the sky”.

Lightning

The lightning is basic. It’s a directional light positioned in zenit above the world directed downwards, acting like a sun.

Helicopter

Next was the fun part, the actual helicopter simulation. We started off by trying to find a “free” 3D object model (in .obj format) to use as the helicopter. We wanted an Apache AH-64 helicopter model and after some searching we found a suitable model.

We cut the model into three pieces, using Blender. We wanted to handle the body, main rotor and back rotor separately to be able to animate the rotors rotating movement. This mainly to be able to animate rotation of the rotors separately and to be able to apply different textures to the different parts.

All the parts were loaded separately, but put and rendered together at the correct relative positions. Finally we made the rotors rotate at different static speeds.

To make the helicopter move, we created functions to manipulate its position. We connected different keyboard key presses to these functions, to control the helicopter.

We used the following key controls setup:

- **W** - Forward
- **S** - Backward
- **A** - Strafe left
- **D** - Strafe right
- **Q** - Upwards
- **E** - Downwards
- **Arrow Up** - Tilt down
- **Arrow Down** - Tilt up
- **Arrow Left** - Turn left
- **Arrow Right** - Turn right

The helicopter is flying by default, that is that it hovers at the current position when no input is received. When pressing keys, the user changes the speed vector (one speed in each direction x, y, z) of the helicopter by a speed increase step (acceleration) until it reaches a given max speed. The speed is slowly decelerating over time down to 0 if no input is given. That is, the helicopter will keep moving for a while and won’t stop instantly when the keys are released. We made this because without it, the helicopter just seemed dull and boring and the controls and feeling of the helicopter movement is everything in this kind of project.

Trees

To make the landscape less dull we added some tree models on the ground. To place them on the correct height we used the same function as in the collision detection described later on.

Collision detection

To prevent the helicopter from flying through the ground we needed to detect collisions with the ground. Since we knew that there wouldn't be any other objects to collide with we could simplify the collision detection to a one-dimensional problem. The height of the helicopter and the height of the ground is compared to detect collisions. We calculated the height of helicopter in world coordinates and created a function to calculate the height at any given (x,z) point on the ground. The function finds the quad matching the world (x, z) coordinates, then it finds in what triangle the point is situated. Finally it uses the triangle's plane equation, created sing the triangle normal, to calculate and get the y coordinate matching the given (x, z) pair.

Problems

Since we are no graphics artists the textures imposed some problems. The seams in the skybox was really ugly and we had to try to mitigate these in The Gimp. It worked out fine thou and the end results were better then expected.

We also had some strange problems with the editing and saving of an image we wanted to use as a heightmap. When loading the image the performance dropped dramatically to a point where the helicopter was completely unflyable despite the fact that the heightmap image was the same as we tested with, just opened and save in the gimp. This made us use an already finished heightmap image.

We also experienced bad performance with huge heightmaps. Initially we wanted to generate a heightmap that matched itself on the edges so it could be repeated "infinitely". This would have needed a lot of more optimisation of the handling and rendering of the ground to work so we got stuck with a smaller, limited world.

We are no masters of the C language and we probably should've left the safe lab environment for an environment where we are more comfortable. Many things that we had a clue about how to implement, took much longer time to implement than expected. Mainly because we are used to higher level languages than C and this made things more complicated for us. The overall program structure would probably increase a bit as well by changing to an object oriented language.

Conclusions

Our ambitions were quite huge in the beginning, wanting to implement nearly everything in the “might do” section. Although, everything took a lot more time than we expected and we contributed in three other different projects in other courses simultaneously.

We managed to implement everything on the “will do” list. We have a textured terrain to fly over, generated using a heightmap image. We have a skybox with a rather good looking texture. We do check for collisions and the program will exit if the helicopter is colliding with the ground. The helicopter is rendered using an imported (.obj) file and the blades are rotating. The camera is at all times situated behind and above the helicopter, as seen in third-person perspective. Although rather basic, we also have a sun light source.

We didn’t implement all of the “might do” features. Although, we did populate the ground with some objects (trees), but didn’t have the time to really make the world interesting and more visually beautiful.

A lot could be added to this, especially if one were to make a game out of it. One thing to add would be using bounding boxes in the collisions detection. We would also like to have a more vivid camera, slowly turning in the direction of the helicopter when moving. It would also be nice to animate the helicopter and tilt it some, relative to the camera, when turning.

At last, it feels after all quite cool to have created a 3D-application that looks and feels a lot like a simulator. It was a very interesting, fun project and we are starting to get even more impressed of all the new 3D games now that we know how much effort that is behind it.