README

Graeme Corrin

I have created a 3d kart racer for my final project.

Parts of the game

Loadup
When the game loads up you find yourself at a screen containing a picture, and flashing text reading "press start." If you hit esc, spacebar, or the mouse, it will go to the main menu.

Main Menu
The main menu has three buttons. One to start the game, one to show you the controls, and one to exit. Starting the game takes you to character select.

Controls
The controls screen simply shows you the controls of the game, and has a back button.

Character Select
This scene allows you to choose the kart you want to race with (using picking). Once the kart you want is selected, there is a start button to start the game.

The Game
The game is a simple time trial. It counts down from 3, then you begin and race around the track and avoid a few rocks. Once you cross the finish line it will take you to the end screen.

The End
The end screen tells you the time you got in your race. Hitting esc, spacebar, or hitting the mouse will take you back to the main menu.

Required Framework

3D Objects
There are 3D objects all throughout the game. The karts, rocks, pillars, and the Character Select floor are all .obj files that I created and then loaded into the game. The Track itself is all modelled using opengl commands.

3D Camera
The camera in the character select moves smoothly between selected karts. While in game you have the choice of three camera positions, standard, first person, and rear view mirror.

Interactivity
You control the kart with the arrow keys while in game.

Lighting and shading
In the character select, there is dim lighting in the scene, with a spotlight shining on the currently

selected kart. In game, the world is light up, until you enter the dark tunnel, then it becomes much darker.

Picking
Picking is used in character select to pick your kart. It uses the hit boxes (spheres) for the karts, so you can click near a kart and it will still register. The ray picking algorithm is robust enough that if an object were behind the camera (along the ray) it would not be selected, and it is set up so that if multiple objects were in the same line, only the closest one would be selected. However, the scene is relatively simple, so not all of this may be apparent.

Texturing
Texturing is used on background images for the menu, as well as all of the buttons, the controls page, and the grass and road texture on the track while in game.

On-screen control panel
While in game, the on screen control panel consists of a timer in the top left corner, and a minimap showing your position in the bottom right. In addition there is a counter that counts down in the center of the screen before you start. The same method used for this control panel is used for the "play" button in the character select scene.

Gameplay
The gameplay consists of racing around a short circuit while being timed. There are three karts to choose from, each with different top speeds and accelerations. The kart can move forward, as well as brake/reverse. There are rocks on one portion of the track, and if you hit one, the rock will be destroyed, and your speed will be halved.

Feature options

Collision detection
There are two types of collision detection in the game, one for the rocks, and one for the borders of the track. The karts and rocks have hit spheres which are used for hit detection. For the borders, a ray-plane intersection is used to see if the kart would hit the border if it kept moving forward, its potion is then adjusted accordingly.

Animation
There is a lot of simple animation throughout the game. In the character select the selected kart rotates and the camera moves smoothly between karts when picking. In game, the kart and the wheels start to rotate when you are turning, and rotate back when you move straight. The wheels also spin at a rate proportional to the speed of the kart. And the scene smoothly fades in and out when you pause and un-pause.

Particles
I have a very simple particle system that shoots out little yellow squares (that are supposed to be sparks) when the kart scrapes along a wall.

Algorithms and data structures

The Kart

The most interesting parts of the code are with the movement of the kart itself. The kart is an object, with a top_speed, acceleration, turning, position, and direction vector. When the up arrow is down, the kart's current speed increases proportional to its acceleration without exceeding it's top speed. When the down arrow is pressed, it decelerates (or accelerates in the reverse direction). When neither key is down, the current speed approaches 0 and the karts comes to rest. On every frame the kart is moved forward by taking its position, and adding its direction vector multiplied by a factor proportional to its current speed (which can be positive or negative). When the left and right arrow keys are hit, the direction vector is rotated, along with the orientation of the kart.

Collision Detection

Collision detection is another interesting algorithm. The hit detection with rocks is very simple, its sphere to sphere collision detection. With the borders, before the kart's position is updated, a ray-plane intersection takes place using the karts direction vector and looping through all planes. All planes have their plane equation, as well as their 4 corner points. The initial ray-plane hit detection finds out if the ray intersects with the whole plane, then checks to see if the intersection point is between the four vertices (so its actually hitting the physical border of the track). If the t value is such that the kart would hit the plane when it's position is updated, then an adjustment is made to its current position before it gets updated. Any other t vale is ignored.

Because all of my hit detection is based on values stored in the objects, and not on just knowing their permanent positions in space, it is then easy to create any kind of track simply by placing objects, and the hit detection will already work.

Originally, I was going to make it so that the direction vector of the kart was rotated if it hit the plane, based on the direction vector and its relation to the planes normal vector. I had it working if the kart approached the plane from 1 of the 4 quadrants. Then I got it working for all angles on a plane that was aligned with an axis. But after testing how this sort of thing is handled in Diddy Kong Racing, and Mario Kart 64, I learned that the kart doesn't need to (and/or shouldn't) be rotated, the direction it moves just has to be adjusted. Which gets very much the same affect (either way you cant go through planes, but you can drive along them).

Optimization

Wherever I could I tried to optimize my code. There are the obvious things, like loading all textures and .objs once and not on every frame. But also, when the karts are created, certain trig values are precomputed, so that when the kart needs to rotate, it only needs to do a simple matrix multiplication with the direction vector using the precomputed trig values, rather than computing them every time the player turns.

How to

The kart is controlled with the arrow keys.
Menus and Kart Select is done with the mouse
Numbers 1, 2, and 3 change camera in game
Spacebar pauses the game

Sources

This game was inspired by Mario Kart 64 and Diddy Kong Racing. I used both games to make decisions on how the kart should control.

My image loader was taken from here:
http://www.google.ca/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CDIQFjAA&url=http%3A%2F%2Fnothings.org%2Fstb_image.c&ei=FZJeUYq0O6TKiwLV7IDgBg&usg=AFQjCNEWZ9sD31gfcrLqz93QjHSCEk8QLA&bvm=bv.44770516,d.cGE&cad=rja

The sky in the background of my title screen art is from:
http://roadstories.ca/big-sky-in-southeastern-alberta/

The road texture I used is from:
http://bgfons.com/download/407

The grass texture I used is from:
http://www.videotutorialsrock.com/opengl_tutorial/crab_pong_part/home.php

I followed through the pseudo code of this tutorial while writing my picking algorithm. There is source code at the end of the tutorial, but I did not use it (my code was better anyways).
http://schabby.de/picking-opengl-ray-tracing/

I learned about lighting from here:
http://www.glprogramming.com/red/chapter05.html

My code for particles was initially taken from here. In this case I actually used the source code in the tutorial, learned what it did and modified it, stripping out the materials, changing size, position, and colour, and making the particles only generate when the kart hits the wall. But the initial code was directly from this site:
http://www.swiftless.com/tutorials/opengl/particles.html

Other than that, everything either came from the lecture slides, peoples posts on piazza, or from my own head.