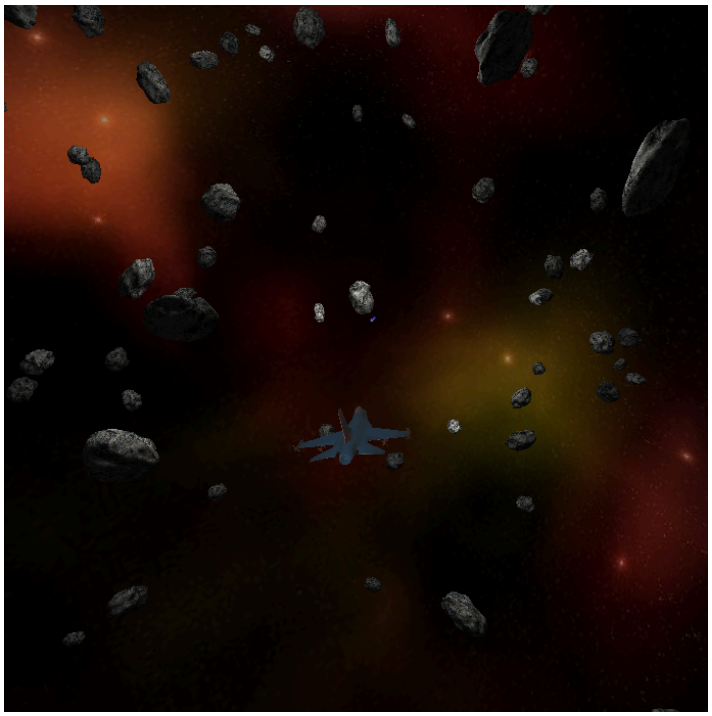


Rerace

by Keith Thompson and Andrew Wagner

Friday, December 11, 2009

Table of Contents

Executive Summary

Game Concept

Rerace is a racing game with a twist. It combines a 2-axis racing game and a 3-axis flying and fighting game. Gameplay happens in two different stages. The game begins with you racing a car along a track against other opponents. At the end of the race you are transported back in time to the beginning of the race. However now you are controlling a flying fighter ship called a “Guardian”. The goal of this stage is to sabotage the opponent racers from the past in order to ensure that your past self wins the race. You are also responsible for defending your past self from the other Guardians who have traveled back in time. If at any point your racer is obstructed, a warning will come up and you will be required to switch back to controlling your racer to avoid the obstacles and return back to your past path. Once leaving your old path you will be required to return before you can switch back to the fighter. The first past player to cross the finish line wins the game.

Development

The game is being developed in C++ using openGL for the interface. GLUT is also being used for creating the window. All game mechanics and drawing is being developed from scratch except for a framework for loading PNG images and OBJ files.

Progress

The basic architecture of the game has been developed including rudimentary physics and environment drawing. There is no gameplay at the moment but the user can drive the racer around in two dimensions and also switch to the Guardian to fly around in three dimensions. The ships are controlled by accelerating in the forward or backward direction and then friction is applied over time to slow the movement.

Next

The next steps are to draw a race track and to implement the physics for collisions with other players and obstacles. Once that is complete other racers need to be implement with an artificial intelligence.

Links

Public Code Repository:

Game Details

Definitions

Stage 1
Racing stage where the user tries to go around a track as fast as possible.

Stage 2
Fighting stage where user tries to influence the outcome of the past race to help themselves win.

Player
A collection of a specific racer and specific guardian controlled by a single user or artificial intelligence.

Racer
The 2D player that races for the first stage.

Guardian
The 3D player that guards the racer and attacks opponent players.

Controls

General Controls

Switch Ship Control	E

Racer Movement

Move Forward	W
Move Backward	S
Turn Left	A
Turn Right	D

Guardian Movement

Move Forward	W
Move Backward	S
Look	Mouse movement

Design Overview

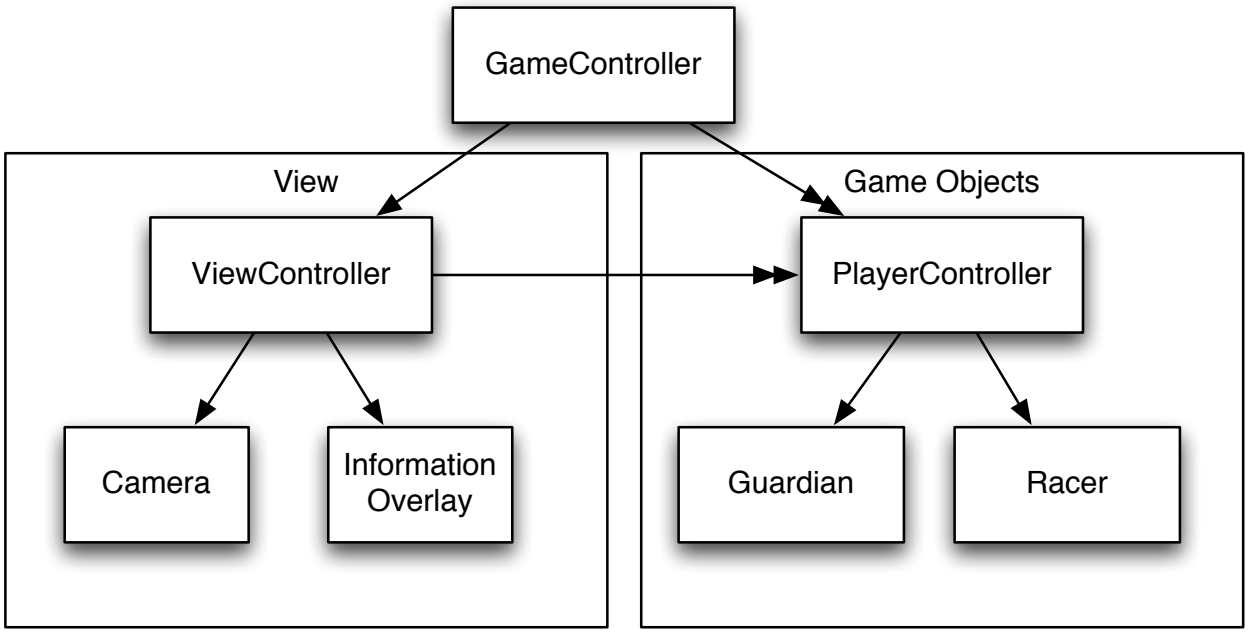
Graphics Techniques Used

- OpenGl Setup
- Glut full screen and windowed
- Mouse Interaction
- Keyboard Interaction
- Matrix Transformations
- 3D Camera Movement
- Texturing
- Lighting
- 3D Model Integration
- Skybox Implementation
- Display Lists

Architecture

Overview

The program is split up between object control and view control. The view control includes a camera object to provide an interface for controlling the camera as well as an information overlay object to control the heads up display. The game objects section includes all objects that are a part of the gameplay.



GameController
Controls each frame of the game. It tells the view controller to draw as well as all the individual players to draw and move. The game controller also receives all input signals from the opengl main loop and calls the necessary functions on the player objects in order to control movement, rotation, and all other interaction with the game.

ViewController
Draws the environment and provides an interface to the camera. It also calls the commands to actually draw the camera and information overlay. Finally it tells the camera to move to the player the user is currently following.

Camera
Object wrapper for controlling camera movement. This handles all of the rotations and translations of the projection view matrix in order to give the illusion of the camera. It allows for moving the camera to any 3D position as well as looking in the direction of a any 3D vector. All movement is created by changing to the projection matrix, setting it to perspective with gluPerpective, and then using the gluLookAt function with the vector stored in the Guardian or Racer.

InformationOverlay
Will handle all information displayed to the user such as lap count and current race position. It is setup to draw 2D text and objects over the 3D scene. Next textures need to be loaded in to draw the letters and numbers necessary for the information display.

PlayerController
Is a proxy to the underlying racer and guardian classes. It allows any other objects to interact with those classes in a unified way in regards to drawing. It also will tell the appropriate class to rotate or move and allows access to information such as their current position and heading. It redirects the commands based on which ship the user is currently controlling. Therefore when the user press “e” to switch ships, it starts redirecting the commands to the other ship.

Racer
Controls 2-axis movement of racer object. It stores a heading vector as well as a speed vector in every direction. When the user presses the key to move forward an acceleration is used to increase the magnitude of the speed vector in the direction the racer is currently facing. A friction is applied each frame to the magnitude of the speed so as to not change the direction of the racer and only slow it down.

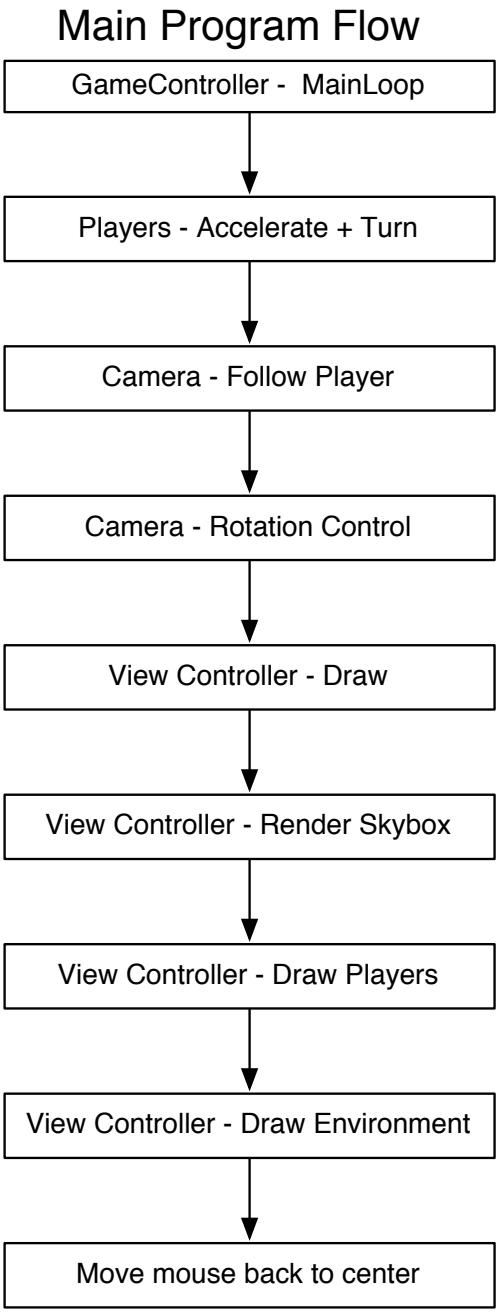
Guardian
Controls 3-axis movement of guardian object. It stores a heading vector, up vector, and right vector. The camera accesses these vectors when following this object so that it can point in the correct direction with the correct up direction allowing the player to fly upside down and in any direction. It also keeps a speed magnitude and always move the player at that speed in the direction of the heading vector. A friction is applied during every frame so that the speed magnitude tends towards 0.

Currently the camera movement in 3D works perfectly but the model of the guardian being drawn is not always in the correct orientation. There seems to be a problem

when using the gluLookat function most likely with the up vector being used. Everything works correctly when looking close to forward but as the player looks further away from forward the model starts to rotate in strange ways. This however is completely independent of the camera and the movement of the guardian which both work correctly. This is something that needs to be fixed in the future.

Program Flow

The program flow is controlled by the GameController main loop. It begins by calling the necessary functions for the players to move with the heading and other values generated from the last loop. Then it moves the camera to that new position. After everything has been moved the user input is then applied to the different game objects. This is done using the keyboard input function and the mouse move function. Afterwords the view controller is told to draw. This causes the rendering of the sky box and then the players. Finally it draws the rest of the environment and then resets the mouse back to the center. From there the whole process is repeated for the next frame.



Utility Functions

Overview

In order to do repetitive tasks we setup groups of utility functions to be used at any point in the code.

VectorMath

Performs basic vector math for rotating vectors during mouse movement and keyboard presses to control player.

Primitives

Draws basic shapes. Used only for debugging.

PNGTexture

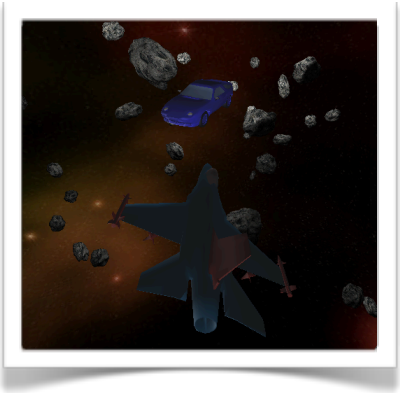
Used for creating a texture out of a png image.

DrawingObjects

Used to generate display lists for all main objects for the game including the skybox, racer, guardian, and astroids. Everything is drawn around the origin and transformations are used outside the functions for scaling, position, and rotation.

glm
Used for loading and manipulating models from OBJ files.

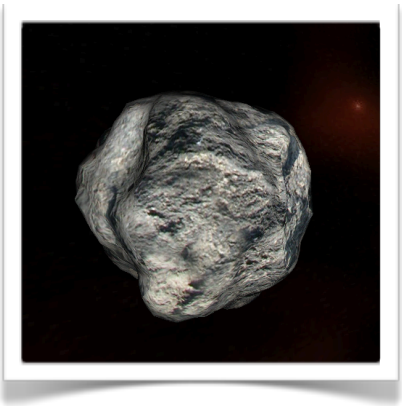
Modeling



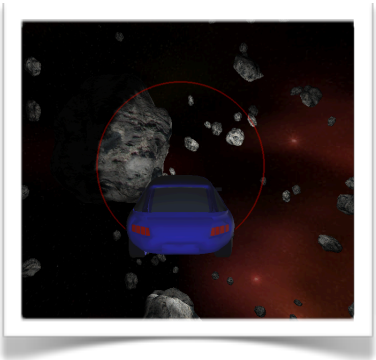
Models were created using a modified version of Nate Robins' C Wavefront OBJ model file format reader, writer, and manipulator. The code was modified to be implemented in C++, and altered to allow for RAW images to be loaded in as textures for models. Placeholder racer ("porsche.obj") and Guardian ("f-16.obj") models were also used from the same source materials. Asteroids were also implemented using the same OBJ loader. All models implemented in the rerace game are stored in display lists for greater efficiency. The asteroid model was then randomly translated and rotated 150 times to create the games asteroid belt world the user interacts with.

Texturing

All textures were stored in the RAW file format for simple loading and reading in C++ and OpenGL. During creation of all textures, the texture minifying and and texture magnifying functions were both set to GL_NEAREST; and the wrap parameters were set to GL_CLAMP in order to eliminate edge boundary issues when creating and merging the skybox textures. Additional code was added to the OBJ loader so that the asteroids could be modeled with a rock texture to give them a more realistic appearance.



Lighting



A total of 4 lights were used in the creation of the Rerace game. Each instance of a Racer and Guardian will have one light source, and there are two world light sources to make the asteroid belt visible. The light sources on the racer and guardian are directional spot lights to give a feeling of headlights when the user is flying in space. These spotlights are placed behind the vehicles so that light will be uniformly cast on the racer and guardian, so the player will always be able to see themselves. the two universe lights are omnidirectional and are spread apart to give even lighting to all asteroids. The skybox has ambient lighting coming from it to make it always visible, but give a less harsh appearance compared to when lighting is disabled for it.