Asteroid Blizzard

Eden Zik and Kahlil Oppenheimer
http://edenzik.github.io/asteroid-blizzard

Overview

We created an asteroid dodging game, with a reverse premise. Instead of the goal being to dodge asteroid, the player's objective in "Asteroid Blizzard" is to crash into them in full force. The more crashes a player achieves in a 3D world, the more points they earn. This game uses the standard WASD navigation keys, along with the spacebar to accelerate and "b" to decelerate.

"Asteroid Blizzard is a space game unlike any other... you are a crazy pilot trying to crash into as many asteroids as possible with your super strong ship. You change direction with either a, w, s, d or the arrow keys. You accelerate with space bar and decelerate with b. The more Asteroids you crash into, the more points you get!"

In this game, we focused most of the efforts on creating a visually elegant world that is focused around the player. The player is an abstract shape that can turn in any direction, and apply thrust. The world consists of a sphere which moves along the player, and clusters of stars (hence "blizzard") which move about the player.

There are several components to this assignment, all of which required some engineering.

Features

Initial Concept - Eden/Kahlil

The concept of this assignment initially involved a plane on which all of the action took place. Inspired by resposities seen online, such as this, we decided to model the world totally in 3D. The idea of this is more around the concept of an infinite 3D world, which is a theme we found interesting.

World Layout - Eden

The layout of this world is spherical, with the player in the center. The trick of the model was to have a sphere with an internal texture, and have the entire sphere move along with the player. This gives the illusion of an infinite space, which can be explored endlessly.

Star Blizzard - Eden/Kahlil

We used oncoming points to model stars, which auto generate as the player moves around the scene. Each star is a white rectangle, given an initial velocity. It is called a star forge, and is based on a repository we found. We modified the code to generate the stars dynamically around the avatar as the avatar navigates through space.

Avatar Basics - Eden/Kahlil

The avatar is a simple convex mesh composed of several points. It is designed to allow the player to fly while still integrating well with the physics engine in the game.

Movement - Eden/Kahlil

We used the keyboard binding “switch statement” demonstrated in class to create unique bindings for each key, enabling us to move around the scene. Each key-press modifies state of the avatar to indicate whether or not the avatar is moving in that direction.

At each render step, the avatar is moved in the directions of the keys that are currently pressed down, and accelerates or brakes if either of those keys are pressed as well. Lastly, the linear velocity of the avatar is actually dynamically modified separate from the acceleration of the avatar such that the avatar always adjusts to move in the direction it points in.

Asteroids - Kahlil

Asteroids were are randomly generated dynamically. Starting position is randomly correlated around the avatar’s current position. Initial speed, size, and color of the asteroids are all randomly generated as well. The number of asteroids and speed/size of the asteroids increase as the game goes on for longer.

Physics - Kahlil

Rather than implementing our own naive physics mechanisms, we used an off the shelf library for modeling collisions called PhysiJS. We configured the physics engine to handle avatar acceleration, deceleration, and collisions with asteroids.

Sound - Kahlil

We added background music as well as collision sound effects randomly sampled from a number of pre-selected sound clips.

Camera - Kahlil

The camera is a “follow camera,” setup to remained focused on the ship with a configurable delayed in its movement when following the avatar (both translationally and rotationally). The camera follow delay creates a realistic feel to flying the ship through space.

Lights - Kahlil

We added a combination of ambient light and directional light (from the sun). We have the directional light positioned to always be shining from the direction of the sun.

Score Keeping - Eden

Upon a collision, score is added to the player indicated in the top right.

Reflective spaceship - Kahlil

The reflections on the spaceship are generated dynamically by a cube camera. Initially, having the cube camera update every render step caused our frame rate to drop to 20 fps on my laptop. We added a configurable parameter to determine after how many render steps the cube camera should update. We brought the frame rate up to 50 fps on my laptop while still maintaining fairly realistic dynamic reflections.

Background Earth and Sun - Eden

We added the earth and the sun to always appear far off in the background. The spaceship can never actually approach them, but their position remains constant relative to the spaceship. When the spaceship moves, the sun and the earth move with it giving the illusion of constantly being far away.

Custom Shaders - Eden

We added custom shaders to the sun to add some cool visual effects. The shader simulates the sun as a fireball with a rippling effect around its surface. Our sun was inspired by examples on http://stemkoski.github.io/Three.js, who had a lot of very cool use cases for Shaders. The ripple effect in the shader are modulated by a timer.

Landing Page - Eden

A landing page to showcase our work
The landing page contains a link to our repository as well.