Author: Erik Gui and Alex Nisnevich
Login: cs184-cx and cs184-dp
SID: 21178740 and 21323554

CS184 Final Project: JezzBall 3D

1 Project Description

For the CS184 final project, we propose to create a 3D version of the classic video game JezzBall. The original JezzBall *(see Figure 1)* is a 1992 game in which red-and-white colored balls bounce about in a rectangular field of play. The player advances to the next level by drawing walls to contain the balls and partitioning the balls in progressively smaller spaces until 75% of the spaces are contained. The player loses a life whenever a ball hits a wall that is being created. In our 3D version of the game, instead of use a plane as the field of play, we will create a cube for the player to partition by drawing planes. The game will allow the player to rotate the field of play arbitrarily.

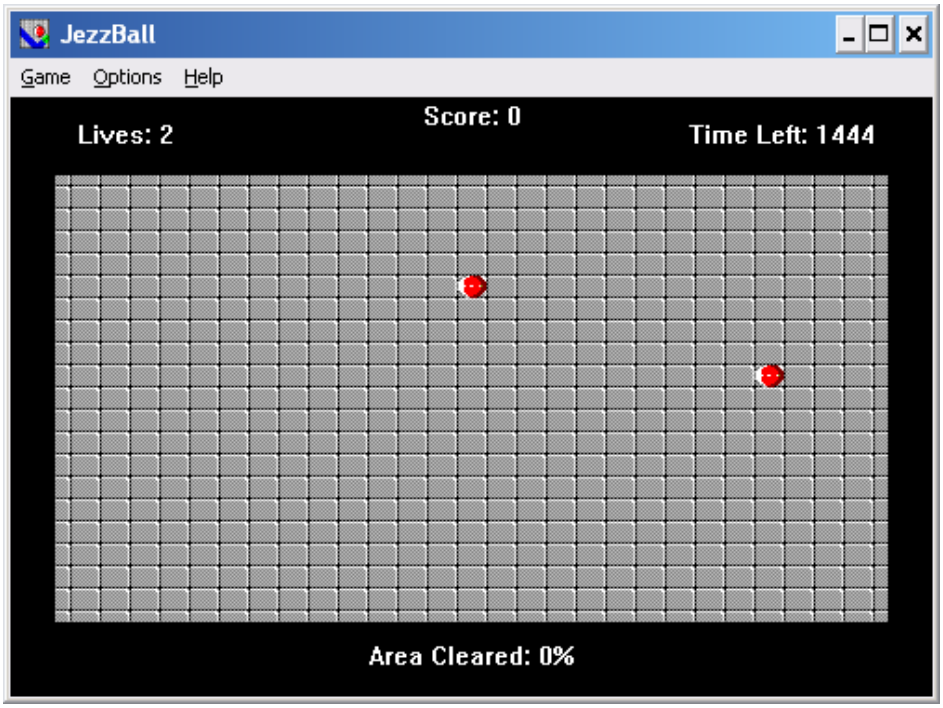

Figure 1: JezzBall

2 Project Challenges Overview

This is an exciting and challenging project, as we have to deal with the physics of the game and also various aspects of player interaction and control. Implementation challenges will include realistic collisions between the balls and the walls (as well as each other), creating transparent objects so that the balls can be seen behind walls, correctly calculating the volume that the balls can and cannot reach, and designing a user-friendly 3-dimensional control scheme. The game will be implemented

using OpenGL and a portion of the rendering pipeline that was established in the course of the semester.

3 Detailed Challenge Breakdown

1. **Realistic Collision Physics:** To detect collisions between two balls, we test if the balls' centers are closer together than the sum of their radii. When a collision occurs, we use the algorithm developed by Thomas Smid and found at `http://www.plasmaphysics.org.uk/programs/coll3d_cpp.htm` to determine the resulting velocities of the two balls. Ball-to-wall collisions are implemented simply by testing whether the ball's center point $\pm$ its radius exceeds the boundaries of the wall.

2. **Intuitive Control Scheme:** To draw planes on a 3D surface, we need to implement a control scheme that would be intuitive to use in an arbitrary 3D orientation. The cursor can be drawn as a long 3D bar that projects from one surface of the cube to its opposite surface *(see Figure 3)*. The orientation of the bar will determine which direction the wall will be drawn, and the player will be able to right-click to select orientation and left-click to draw the wall. In order to control the position of the wall, we use the `gluUnProject` function to track the mouse position in 3D, by determining where the cursor lands on the cube faces.

3. **Calculating Unreachable Volume:** The winning condition for each level is to clear 75% of the play field - that is, to partition the play field in such a way that at least 75% of it is unreachable to the balls. Assuming that a XY-planes, b YZ-planes, and c ZX-planes have been drawn by the player, there are a total of $(a+1)(b+1)(c+1)$ boxes that must be tested to see if any balls are located within them, which should not be time-intensive even for a naive algorithm. However, the situation is complicated by the fact that, if a ball hits one side of a plane that is being drawn, only half of the plane is drawn, which means that not all planes will go from one end of the play field to the other end. It is thus nontrivial to determine how the play field is partitioned. One possible solution is to create a new class that keeps track of all boxes that subdivide the play field and handles all relevant computations, such as determining which new boxes are created when a new wall is drawn and keeping track of which boxes are accessible to balls.

4. **Simple and Interesting Game UI, Effects, and Sound:** We can use various open source Game UI libraries such as `http://www.bramstein.com/projects/gui/` and `http://www.cs.unc.edu/~rademach/glui/` to create our game menu. Game effects, sound effects, and background music will be added to the game to make the game more interactive. For adding game sounds, we could use some audio mixer libraries such as `http://www.libsdl.org/` specifically the SDL mixer found here: `http://www.libsdl.org/projects/SDL_mixer` . As for game effects, animations can be added to certain events in the game. Shader effects can also be used to further enhance the game graphics.

Author: Erik Gui and Alex Nisnevich
Login: cs184-cx and cs184-dp
SID: 21178740 and 21323554

4 Project Team Information

Two students will complete this project. It should be a reasonable project to implement in the remaining weeks of the semester.

5 Current Progress Screenshots and Descriptions

This week we implemented many fundamental portions of the game, including defining some basic classes, setting up a play field, implementing the collision physics of the game (both ball-to-ball and ball-to-wall), and tracking the position of the mouse in 3D. We also implemented anti-aliasing for lines.

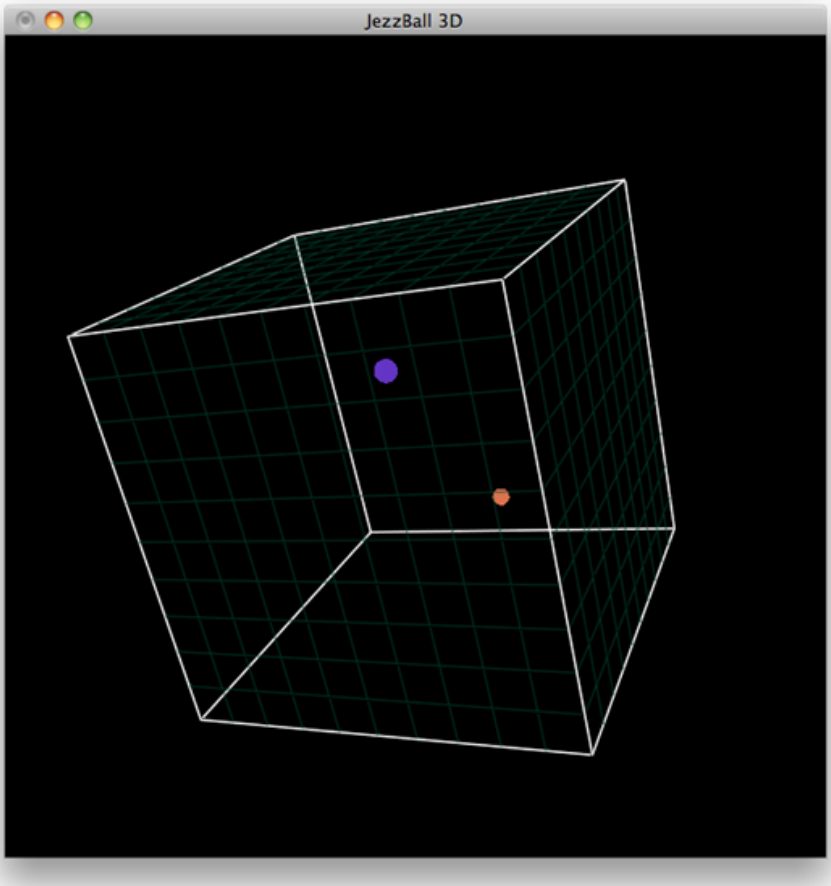

Figure 2: This screenshot shows the setup of the game, along with two balls in the playfield. The balls bounce about the field with constant velocity and realistic collision physics.

Author: Erik Gui and Alex Nisnevich
Login: cs184-cx and cs184-dp
SID: 21178740 and 21323554

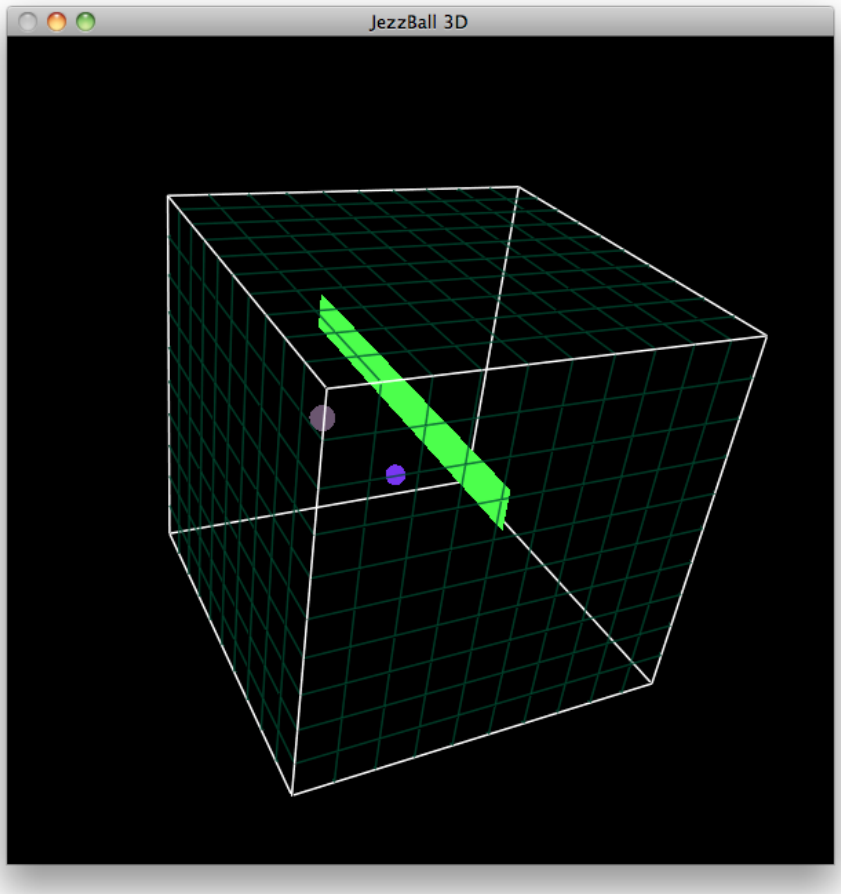

Figure 3: This screenshot shows the mouse tracking, as the green rectangle follows the location of the mouse as projected onto the cube surface.