# Cityscape Golf: Final Report

Cal Pierog, Ryan Riegel, Paul Scott

April 13, 2004

## 1 Overview

We essentially built the golf game that our professor specified, and set it in a cityscape-themed course. We didn't have any major problems with the design requirements, but we did face our share of hard-to-find bugs (one of them even evolved into an interesting feature!). As in any golf game, the player's goal is to get the ball from the tee to the hole in as few strokes as possible.

## 2 Gameplay

### 2.1 General

The game occurs in a rendered 3D playfield containing the ball, club, various obstacles, and a hole. Whenever the ball is stopped, the player can send it flying by hitting it with the club. If the player gets the ball to the hole, the player wins and the game is over.

### 2.2 Player Controls

#### 2.2.1 Camera Adjustment

The player may always adjust the camera's viewing angle by left-clicking on the game window and dragging. Also, the player can always adjust the camera's zoom with `+` and `-`. By default, the camera stays focused on the ball, but the player can press `b` to toggle between tracking the ball and free-flight. When in free-flight mode, the player can use `w`, `a`, `s`, `d`, `c`, and `v` (i.e., Half-Life movement controls) to adjust the camera's position.

#### 2.2.2 Hitting the Ball

When the ball is stopped and the camera is tracking the ball, the golf club appears between the camera and the ball. The player aims by adjusting the camera angle; the ball is hit away from the camera, but always at a small fixed angle relative to level ground. This is to reflect the fact that you must stand upright to properly use a golf club, and the ball should be at about the bottom of your swing. The effect is that the player cannot hit the ball into the ground or straight up, etc.

The player starts to swing by pressing the space bar. This causes a meter to appear on the right side of the screen which indicates swing strength. This meter repeatedly increases and then decreases until the player presses the space bar again, and then a velocity proportional to the height of the meter is imparted upon the ball.

### 2.2.3 Debugging Controls

There are several commands left in by the developers for the purposes of collecting debug information. These are as follows:

| | |
|---|---|
| `z` | Toggle display of axes |
| `p` | Cycle parsing verbosity |
| `o` | Toggle collision/rolling/stopped notifications |
| `n` | Output ball's position |
| `N` | Toggle ball position/velocity reporting |
| `l` | Load a file |
| `\` | Toggle triangle adjust mode |
| `[, ]` | Separate/Conjoin triangles (triangle adjust mode only) |

## 2.3 Physics Engine

### 2.3.1 Collisions

As per the specification, the ball can collide with triangle faces, edges, and vertices. Because computer calculation is discrete and ultimately imperfect, we do what we can to make each time step as accurate as possible and we also employ a collision buffer. The collision buffer serves to keep the ball from getting too close to obstructions, and thus circumvents the problem of balls slipping through triangles due to numerical inaccuracy in the professor's recommended method of collision detection.

In response to collisions, the ball's velocity is reflected through the normal of the collided obstruction and then reduced by a damping factor. In the case of triangle faces, the normal is fixed, but for edges and vertices the normal is actually calculated on the spot depending on the position of the ball. This serves to maximize realism.

### 2.3.2 Rolling and Stopping

If the ball is effectively touching some obstruction and its velocity in the normal of the obstruction is very small, then the ball attains a rolling status with that obstruction. Gravity is no longer added in opposition of the obstruction's normal, thus preventing the ball from perpetually bouncing due to the discrete nature of the physics engine. Also, when rolling, friction acts upon the ball to slow it down. As soon as the ball moves away from the obstruction(s) it's rolling on, the ball loses its rolling status and proceeds to act normally.

If the ball is rolling on something and has very small overall velocity, it attains the stopped status. This causes the physics engine to stop processing and wait for the user to hit the ball again. The restriction that the ball must be rolling in order to stop is in place to keep the ball from stopping in mid air (a problem we had on very fast computers).

### 2.4 Graphics

The game uses OpenGL to render the many triangles that make up the course, ball, and club. Lighting and colors are implemented, but no textures.

### 2.5 Course

There is just one course in the game (indeed, adding more course is a possible future enhancement). The course is cityscape-themed, having tall buildings and arches, and being relatively large compared to the ball. There are no sand traps, but it is possible to hit the ball off the edge of the course, resulting in a one stroke penalty and the ball returning to the last position it was stopped. The hole is located at the end of the course opposite the ball (as one would expect); it's red, fairly large, and star-shaped. A triangular "flag" floats above the hole, just in case it wasn't distinctive enough.

### 2.6 Scoring

As in most golf games, the player aims for the least amount of strokes to get the ball in the hole. Each swing counts as one stroke (of course), and there is a one stroke penalty for hitting the ball off the edge of the course. We haven't officially set a par for the course, but it's probably close to five.

When the player gets the ball to the hole, the course undergoes a shattering effect (enacted by automatically shrinking triangles with the `[` command). This feature actually had its origins as a bug, but was soon harnessed as a debugging utility and an impressive grand finale. Who else can claim their golf course explodes upon completion?

## 3 Bugs and Possible Improvements

While the game is playable and fun, it's not bug free. We've spotted the following:

- Rolling is somewhat unreliable because it is based on the ball's velocity rather than it's actual change in position. The velocity (before rolling has taken effect), includes the small amount of gravity added in half a time step, and thus looks artificially larger than it actually is. (This problem was fixed shortly after the code freeze by making rolling only consider actual changes in ball position; the revised version will be used for the class presentation, but will not be submitted.)

- The game behaves differently on very fast and very slow computers because time steps are variable length. The variable length time step was actually introduced to help make the performance on machines of different speeds more comparable, but because it is also used in checking for stopping (and rolling in the demo) by comparing ball positions over the last 50 time steps, very fast computers may be more prone to stop the ball because, on fast computers, the ball doesn't move as far in 50 time steps.

- There are numerous graphics issues involving intersecting and overlapping surfaces. This could have been fixed if the course was reworked so that such surfaces did not exist, but it wasn't urgent and we didn't have the time.

Also, there are a few areas where our game could be improved:

- An octree could have been implemented to allow for bigger, more complicated courses.
- More courses would make the game more interesting and worthwhile.
- The course could also be made more interesting by adding obstacles such as sand traps.
- The graphics could be improved by adding support for textures or more advanced lighting effects. Also, the club could be make to move when you hit the ball and the ball could be made to appear as if rolling.
- Golfing sounds and music could have been added.
- A Hall of Fame could be implemented so you can brag to your friends about your birdie, etc.
- The interface could be manipulated so that only the OpenGL was necessary to play. As it is, one must look at the console to see the amount of strokes so far.