Balls on the Tray

2022-2nd Semester

[CSE471] Computer Graphics

Final Report

Student ID.	20161105	20161190
Name	Soogeun Park	Minjae Lee
Project Advisor	Ilwoo Lyu	

FIRST IN CHANGE

I. Introduction

1) Motivation

We observed the people's enjoyment of putting the ball into the basket. We thought that a game that can stimulate this enjoyment and is easy to manipulate would be a fun game. Based on these ideas, we came up with a game named ‘Balls on the Tray’, in which users shake the tray and put the balls on the tray into the designated basket.

Thus, the goal of this project is to create the game we want ourselves. And we intended to make an enjoyable game and apply the Computer Graphics principles such as object transformation (rotation, translation), coordinate transform pipeline, and collision detection.

2) Specification

To implement this game, we need three types of objects: a ball, a tray, and a basket. In this game, balls spawn above the tray, fall onto it, roll on the inclined tray, and must be dropped into a basket if it lands in it. To represent these movements, we need to calculate the movement of the object. And for the user to recognize that the balls are moving in real-time, new frames must be continuously displayed at very short intervals. In addition, the user should be able to manipulate the tray, and it must provide events to continuously spawn balls and move the basket as the game progresses. We also need a text interface to provide information about the remaining time in the game and the user's score. Lastly, we plan to provide sound effects along with in-game events so that users can feel the visual information in the game more realistically and immerse themselves in the game more.

In summary, this game has a total of 7 requirements: (1) 3D Object rendering, (2) Calculate the ball’s position using physics, (3) Real-time update of the screen, (4) Set the camera viewport, (5) User manipulation & Game event handling, (6) Text interface for displaying information, and (7) Sound effects.

II. Main Subject

1) Approach

The three types of objects used in games have different appearances and functions from other types, while objects of the same type share common appearances and functions. In addition, in the case of balls and baskets, there is not only one object. Especially the balls are continuously spawned during the game, and the number, color, size, etc. are not fixed. Therefore, object rendering can be implemented by defining each type of object as a class with a render function and executing only these render functions at the ‘display’ function of the ‘Viewer’ class that is responsible for displaying the game. On the other hand, the objects needed in the game have a simple shape. Therefore, 3D Object Rendering can be implemented using the basic rendered object function (glutSolidSphere, and glutSolidTorus) provided by the GLUT[1] library and a circle plane using GL_TRIANGLE_FAN. Detailed implementations of baskets and trays are introduced in the next section.

Calculating the movement of a ball is finding the position of the ball in the next frame. The position

of the ball in the next frame is the position of the ball in the previous frame plus the change in position, i.e., velocity. And this velocity changes by the amount of change in velocity, the acceleration. The factors that determine the acceleration considered in this project are the gravitational acceleration, the frictional force of the tray plane, and the elastic modulus of the ball. Another major factor that determines the ball's movement is to detect whether the ball has collided with another object (tray and basket). If the ball hits the tray, the ball bounces up, and when the ball is on the tray, the gravitational acceleration must be decomposed into the tangent direction of the tray and added. And if the ball goes into the basket, it should be kept in the basket without falling. Since all objects used in the game consist of only spheres and circles, collision detection can be calculated only with implicit representations. That is, the calculation of the numerical formula is sufficient.

Real-time updates of the screen can be implemented by continuously executing the ‘display’ function at short intervals. This can be easily implemented using the ‘glutTimerFunc’ function provided by the Glut library.

This game does not require a shift of view. So, if it is found a suitable fixed view to keep track of the game's progress, we do not need to move the camera position. Meanwhile, since the users need to be able to feel the height information, a perspective view is used.

Manipulation through the user's keyboard input can be handled by using ‘glutKeyboardFunc’ and ‘glutSpecialFunc’. When users input a direction key, we bind a function that rotates the tray in the corresponding direction so that the user can rotate the tray. In the game, balls are spawned and contained in the list named ‘balls’, which is a property of the Viewer class, and are all rendered by traversing this list with the display function. Raising events at random and regular intervals is described in the next section.

Since the text interface is not provided by OpenGL[2] by default, it was implemented using the ‘glutBitmapCharacter’ function provided by GLUT. This is implemented as the ‘drawText’ function in utils.py.

Since OpenGL does not support sound-related functions, we needed another module. The module we needed was not playing just one sound but had a function to play several sounds simultaneously and end the sound at the desired time. We also needed a natural, high-quality sound that fits the situation, for a sense of reality. So, we found a Pygame[3] module for liberal sound control. We can read various types of sound files from Python to a module for games and operate them simultaneously. It was also supported to terminate in certain circumstances. And we obtained high-quality sounds from webpage[4] which provides several sounds for game developers. But it was mostly the euro, free sounds were not high quality. One of our team is good at sound editing, so we processed free sounds using a daw program called Cubase[5] for sound mixing and mastering. We processed background, heavy ball bounce, light ball bounce, basket success, basket fail and game end sounds using an equalizer, dynamic compressor, and reverb effect technique. The background sound of the game used the beat[6] that a team member made himself.

2) Implementation

In this section, we will skip the things explained in the previous section.

Ball, Tray, and Basket Classes are defined in the file named 'Models.py'. The rendering of each class object is defined in each ‘render’ method. Using GL_TRIANGLE_FAN, Tray was implemented

by attaching triangles connecting the origin and two adjacent points around the circle. At this time, the number of points at regular intervals taken around the circumference of the circle is defined as sharp (sharpness, the default value is 100). The basket consists of a wall surface made by attaching several tori and a bottom surface implemented in the same way as a tray. The height of the basket means the number of connected tori, and the default is set to 100.

The movement of the ball is implemented in the 'update' method of the Ball class. The acceleration is given by the magnitude of the gravitational acceleration in the y-axis direction (vertical drop motion). And the velocity is updated to the previous value plus the acceleration. This velocity is decomposed into v_n(normal-direction velocity) obtained by cross-product the tray's normal vector, and v_t(tangent-direction velocity) obtained by subtracting vn from velocity. If the ball collides with the tray, v_n is multiplied by the elastic modulus and then flipped, and v_t is multiplied by the friction coefficient. Figure 1 shows the collision conditions between the ball and the tray, which are (1) 'the distance between the center of the ball and the tray plane is smaller than the radius of the ball' and (2) 'the distance between the center of the ball and the center of the tray is smaller than the hypotenuse of the right triangle which has the sides as the radius of the tray and the ball'. The calculated v_n and v_t are added again to create the final velocity, and the new position is obtained by adding it to the previous position.

Figure 1: The collision conditions between the ball and the tray.

However, when the ball collides with the tray, the position of the ball is lower than the tray. And since the height at which the ball bounces is multiplied by the modulus of elasticity, it rises smaller than the previous lowered position. Therefore, when the ball collides with the tray, it is necessary to adjust the position of the ball so that it is placed on the tray by adding the opposite amount as much as the ball went down the tray. This is implemented by adding the normal vector multiplied by the value of ball radius subtracted by the plane distance (normal vector * (ball radius – plane distance)).

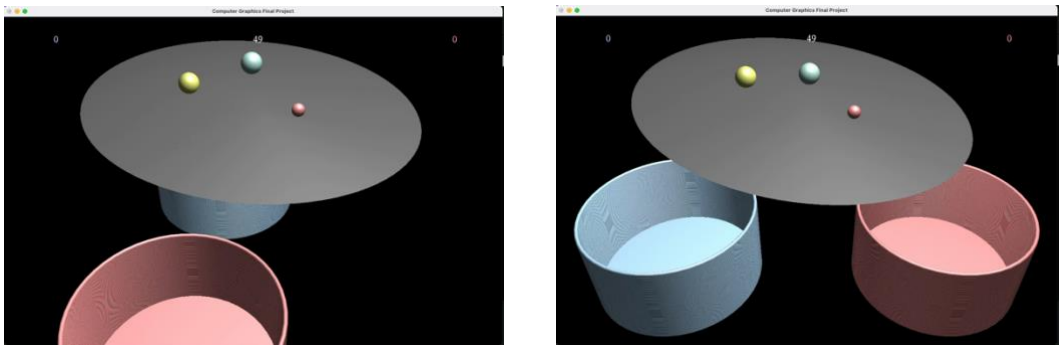

Figure 2: A reproduction sinking ball bug. The left is the bug, and the right is the normal operation.

Whether or not the ball entered the basket is determined by whether the ball collided with the bottom of the basket. Detecting for collisions is the same as for Tray. The Ball class has a property named 'goal_in' which checks if the ball has entered the basket. If it hits the bottom of the basket, goal_in is set to True and these goal_in balls are not rendered, reducing unnecessary computation.

To implement the generation of game events at regular or random intervals, we use the 'threading.Timer()' function. This function executes a callback function at regular intervals. We used this to run the 'tick' function in the Viewer class every 0.5 seconds, which generates random variables p, and q, with arbitrary values between 0 and 1. The 'add_ball' event occurs when p is less than 0.7, and the 'move_basket' event occurs when q is less than 0.2 (which means 70% and 20%, respectively).

We implemented our sound using the Pygame module. We display Pygame using the method, `display.set_caption`. And load the mp3 file using the method 'mixer.Sound' of Pygame. Using the `set_volume` function, we make players hear natural game sounds by considering the appropriate sound size. The background sound is not terminated by using 'self.background.play(-1)'. When the collision between the ball and the tray occurs, heavy ball sound is executed in the case of a ball with a radius greater than 0.5, and light ball sound is executed in the case of a ball with a radius less than 0.5. In addition, if the ball speed is lower than a threshold, the sound is reduced by a `set_volume` function. When a normal ball enters the basket, a success sound is executed, and when a black ball enters the basket, a bomb sound is executed. Finally, when the remaining time becomes equal to 0, we set the game termination sound (window termination) implemented.

III. Conclusion and Discussion

1) Project Results

We worked well on the specifications we planned. The ball, tray, and basket, which are 3D objects, were well rendered. The camera's viewport is also set. We also succeeded in updating the position of the ball according to the timer and time. The tray was rotated through the matrix, and the movement of the ball on the inclined tray was also expressed. It succeeded in expressing the motion of the ball by defining the area of the tray and whether the ball will fall or bounce again. It was also well detected that the ball fell from the randomly changed basket. It also succeeded in implementing a user interface such as showing the remaining time and the player's score. Finally, Pygame was used to set up well so that the appropriate sound could be made in each situation.

The game rules

1. It is a Two-Player game that lasted for 50 seconds.
2. Players can rotate the tray using the key.
   (Player 1: right/left/up/down Player 2: d(right)/a(left)/w(up)/s(down))
3. Players can get points by putting the colored ball in their basket, and the player who has the many points wins.
4. However, if a black ball goes into the basket, it loses points (-3).
   It is also possible to put a black ball in another player's basket.

5. A ball of random size, position, and the color is periodically created above the tray.
6. The location of the basket also changes over random time.
7. Run the code and press any button, then start the game.

We uploaded the demo video on YouTube, and the code and playing demo are also available in the GitHub repository. Here is the link:

- YouTube Link: https://www.youtube.com/watch?v=QdZnr0PjK3M.
- GitHub Link: https://github.com/bwmelon97/balls-on-the-tray

2) Discussion

Is there anything you couldn't solve? If so, why?

We failed to implement some of the properties that we wanted due to time constraints.

1. We don't consider the weight of the ball. The larger ball has a larger weight, so the bigger gravity work. But all balls fall at the same speed. Therefore, collision detection between balls could not be considered.
2. We don't consider light and shadowing. Although we succeeded in implementing simple light, we did not consider light applied according to the position of the ball.
3. We can't address the user interface's font size. We tried to increase the font size to make users comfortably aware of information such as scores and times, but we failed.

Briefly discuss how you can improve your work.

First, Various modes can be added. In the case of snow mode, there is snow on the tray, so a greater friction force is applied to the balls. As a result, the elastic coefficient of the ball also decreases, so balls can't move well. It will require more detailed control to put the ball into the basket. In the case of random obstacle mode, fixed obstacles are attached to the tray. If the ball hits an obstacle, players should be careful because it can bounce unexpectedly. In addition to these modes, there are fire modes in which the ball disappears when it touches the fire, moon modes with different gravitational acceleration, and slippery modes that oil on the trays.

Second, it is possible to improve not only the ball but also the items that randomly give special effects to fall off. For some examples, the muscle item can rotate the tray at a greater rate than other players. The black hole item can exert a new pulling force on the balls toward the player's basket. It is possible to improve not only balls but also items that randomly give falling special effects. Also, fainting items can disable the other person's keyboard for a while.

Third, we can further improve the user interface and 3D object design. For example, adding a game start button or end message at the end of the game. If the first and second improvement points are added, the related interface needs to be added. In addition, 3D objects (balls, trays, and baskets) were simply rendered, and their designs could be upgraded in more detail.

The contribution of each team member.

Soogeun Park

- Propose the project idea
- Implement object rendering, physics engine, event handling, and user interface
- Write the final report and manage the GitHub repository

Minjae Lee

- Improve and expand the project idea
- Implement sound effects, and tune the parameters of the game for user engagement
- Make a presentation (including preparing materials)
- Write the final report and make the demo video

※ References

[1] https://www.opengl.org/resources/libraries/glut/

[2] https://www.opengl.org/

[3] https://www.pygame.org/news

[4] https://www.mewpot.com/search/sound-effects?tag_ids%5B%5D=10618

[5] https://www.steinberg.net/cubase/

[6] https://www.youtube.com/watch?v=e_wGq72Xg2Q