KOÇ UNIVERSITY

College of Engineering

COMP 410 – Computer Graphics

Project Report

BrickBreaker3D

Participant Information:

Osman Kaan Demiröz 30388

07.06.2016

Introduction

Brick Breaker is a very simple and old game in which the user tries to break all the bricks in the level by bouncing a ball on a small rectangular surface and hitting the bricks. The level is constrained by walls and there exist many different types of bricks. According to the type of brick, it may interact differently with the ball, or destroyed bricks may generate bonuses.

The purpose of this project is to implement the Brick Breaker game in a 3D environment. While carrying the 2D structure of the Brick Breaker world to a 3D scene, rectangles become rectangular boxes, and circles become spheres. Creating this simple mapping would be sufficient to create a 3D representation of the game. However, this project is also an opportunity to add a third dimensional element to the structure of the game. In the original Brick Breaker game, blocks reside at the top of the screen, extending in two dimensions, X and Y. The rectangular flat tray at the bottom can move in the X direction to catch the ball and reflect it in different angles. Having a 3D world, the limits of movement can be extended to the third dimension. The ball can now move in the Z direction as well. The blocks at the top of the screen can also have pieces on top of others, creating different levels. Finally, the rectangular box tray can now move in both X and Z directions to catch the ball moving in three dimensions.

With the addition of the third dimension, Brick Breaker 3D is aimed to present a novel and fresh type of experience. To create and emulate the environment, the project uses the Java3D package and the Scene Graph Implementation that is brought with it. The project therefore makes use of Object Oriented Programming.

Background

A typical game of Brick Breaker consists of 3 main parts: The ball, the tray, and the blocks. The outside boundary walls can also be thought of a part, if the implementation does not include its own fixed boundaries. The game must make sure that the interactions between these parts are handled correctly.

The blocks can be added to the environment using a structure called a block matrix. The block matrix is an n-dimensional array, where n is the number of spatial dimensions. The matrix

contains every block in the same order as they will be displayed (row, column …). As the game progresses, new block matrices can be used to represent levels, and blocks can be removed from the block matrix to represent the destroy effects on the blocks, as the ball hits them.

The tray requires a way of getting user interaction. Since the tray is the user's way of hitting the ball, it should get either key presses or mouse interactions. The way it has been done in this project is that the user's mouse position is monitored and the tray follows the mouse accordingly.

Project Specification

The project makes use of Object Oriented Programming via Java3D, since there exist many instances of the same objects in the environment. Java3D provides much ease in this case. It operates on the Scene Graph Implementation of Computer Graphics, rather than the Pipeline Implementation.

As pieces of the Object Oriented Design, all objects inherit from a common base class called ColorObject. This class is a specialized version of the BranchGroup object in Java3D, which is basically a parent node in the Scene Graph. It contains its own TransformGroup element, its Primitive object, and its Appearance components. A subclass of ColorObject is RectangularObject, which is a specialized version whose Primitive object is a Box. The main objects (Ball, Tray, Blocks) themselves are specialized versions of these classes. The Ball is a subclass of ColorObject, while the tray, the blocks and the outer walls are all subclasses of RectangularObject.

Problem Analysis

The novelty of this game to the original Brick Breaker is the implementation of the scene in a 3D atmosphere. This addition includes three main aspects. The first challenge that lies is that the concepts must be adapted to a new programming environment. The programming environment in this case is Java3D. The second key point is the conversion of 2D objects in a regular Brick Breaker game to 3D objects. The third and the final element is the additional game feature that is implemented in this project, which is the extra dimension that comes with the 3D

scene, and that is the Depth element. Allowing the implementation of these three main aspects leads to the completion of this project and thus, to the completion of the game.

Since a 3D API is used, the corresponding methods and usages must be found and added correctly. Many methods in Java3D are similar to the concepts that the Computer Graphics course has taught in OpenGL. However, due to the Object Oriented structure of Java3D, the rendering model is different. Objects are added as nodes of a Scene Graph, and at runtime, the Scene Graph nodes are added one by one and rendered. This provides flexibility over the regular Pipeline Implementation of OpenGL, but is more costly performance-wise. In a regular Java program, certain geometric shapes are already defined such as Rectangle or Circle. Implementing a Brick Breaker in such an environment would therefore be simpler; as the superclasses of the main objects would be much more straightforward. However, since the Scene Graph implementation contains many elements, an object such as the Ball, should include more elements than only the 3D shape. An example to this complexity is the fact that objects are rendered according to the TransformGroups that they are in, and their appearances are modeled according to the Appearance objects that are associated with them. Therefore, a Ball class should contain all of these aspects within itself.

Implementing an originally 2D game in a 3D world brings several minor challenges. All objects that were originally generated with 2D information now need to be coded with 3D information. Boxes need their depth information, and circles need to be added as spheres. This causes the necessity that mere coloring on a surface does not create good visuals. In a 3D world, lighting needs to be taken into account when coloring objects. Therefore shading properties need to be given to objects rather than colors. Another difference of the 3D environment compared to a fixed 2D scene is that there is not a single canvas where the user paints shapes on. There exists a camera through which the user sees the scene, and the camera observes the world. Therefore, the transformations and rotations need to be applied in a different fashion than just placing rotated objects on specific points on a canvas. It is at this point that the Computer Graphics course information comes into play the most.

The last important element in implementing the game in 3D is the addition of the Depth element for the gameplay. While the original game moves in the X and Y directions, now the

objects move also in the Z direction, and the blocks may be placed in different levels. As a demonstration of this property, the current state of the game contains 3 levels of blocks. In addition, the ball moves in a 3D world rather than just in X and Y. To compensate for this added dynamism in the game, the user's tray also needs to be able to move in the Z direction, so that it can catch the Ball when it arrives at the top or at the bottom. Implementing these key features is crucial in successfully completing the project.

Solution & Implementation

The main function of the project is fairly straightforward. A new instance of the game scene is created, and the main application frame is configured. For convenience, a blank cursor has been set for the user. This way, as the user moves their mouse while playing the game, the cursor does not distract the user and only the tray is left as the focus. When the main function terminates, the init() function is called to setup the scene.

In the init() function, the application frame is further configured, and the simple Universe is generated, where the Scene Graph objects will be placed and the rendered objects will be seen. The camera is initialized and positioned over the origin, so that objects placed near the origin will be the focus of the game. The Scene-creation method is also called in the initialization method, and after the scene is created, important interaction points are instantiated. These are the mouse and key interaction listeners, and the game timer. Mouse movement is used for moving the user's tray, and key presses are monitored to grant interface-related functionalities to the user. When the user moves their mouse (either pressed or not) the X and Y coordinates of the user's mouse are transferred to the listener, and after the values are multiplied by a weight scale, the tray is positioned according to the mouse position. The Y coordinate in the world in fact moves the tray closer to the blocks, and this is not something that is desired. Hence, instead of the Y coordinate, the Z coordinate of the tray is modified according to the Y coordinate of the mouse. This creates the effect that the upward and downward motions of the mouse lift the tray above or move it below in the depth dimension. The keyboard listener exists to provide control to the user. Pressing the key "i", the user can center the camera to the original location, resetting any rotation and translation. Pressing "p", the user can pause and resume the game, and pressing "q", the user can quit the game. An additional feature added later to the project is that by

pressing Alt+Enter, the user can enter full screen mode, removing the outer bars of the application window. The timer is also necessary so that the ball can move smoothly. As the user begins the game, the timer starts, and at each time step, the ball moves just a little bit, creating a smooth motion effect. After these functionalities are added in the initialization method, a default orbital camera feature is added which allows the user to rotate the view by holding the mouse and moving, and zoom in-out by scrolling the mouse wheel. The user can also move the camera in any direction by holding the right click and moving the mouse, but this provides no further functionality, as there is nothing outside the given boundaries. At the end of the init() method, the Scene Graph is added to the universe, and the objects are displayed to the user.

While the Scene is being created, important factors are the objects, and the lights. All things that are to be added to the scene have to be added to the Scene Graph in the end. There are two light sources in the game. One is an ambient light, and one is a directional light. These two lights allow better coloring of the shapes. BoundingSpheres are created for the lights as well. These spheres show the interval in which the lights will shine. Once the lights are created, they are added to the main Scene Graph. Later, the objects are created and added to the game. These objects include the outer boundary walls, the Block Matrix, the user's Tray, and the Ball. While the objects are being created, Appearance instances are created which contain information about their material properties. This way, the objects' colors are determined. Once all objects are created, they are again added to the Scene Graph. Finally, a collision listener is created to handle collisions with the ball, and the collision listener is also added to the Scene Graph. The resulting Scene Graph is compiled and the objects are added to the user's display.

A hierarchy has been created to easily create and classify different objects. The main superclass in this hierarchy is the ColorObject class. This class has been defined to contain all object properties in a single class. It extends the BranchGroup class which acts as a node in the Scene Graph and can have its own children nodes. A ColorObject's children nodes in this project are the TransformGroup that allows the object to be translated and rotated, the Primitive that is basically the 3D shape of the object, and the Appearance which contains information about material and the color of the object. The main objects in the game all extend this class, and have their own special functions added to the template. The Ball class uses a sphere as the Primitive object, while the rest of the objects, the Block, the Tray, and the Walls use Box'es as the

Primitive objects. A BlockMatrix class is also created to better handle the mass of Blocks in the game. Creating a BlockMatrix creates a general TransformGroup in which all blocks can be moved together to a position. The BlockMatrix creates all the blocks as its children and adds them to the Scene Graph. It also has the ability to remove its children, so that when the ball touches one of the blocks, the BlockMatrix can remove the proper block to demonstrate the destroy action.

The CollisionListener is one of the key points in the game since without proper collision detection a Brick Breaker game cannot be created. While instantiating the CollisionDetector, a "Bounds" object needs to be passed, and this is the interval in which collisions need to be checked. In this case, the bounds of the Ball are passed, since only the collisions with the ball are important in the game. When initialized, the WakeupOnCollisionEntry and WakeupOnCollisionExit methods of the Java3D library are assigned to check for the ball. Throughout the game, when the ball collides with any "collidable" object, (which all objects in the game are marked as), the onCollisionStart and onCollisionEnd callback functions are launched according to the action that occurred. When a collision starts, the Ball's motion needs to be updated according to the direction in which collision occurred. Because of this, the sides of the collided object are checked. When the ball collides with a block, for example, if the collided face is the left face of the block, then the ball should reflect to the left. Implementing this behavior via these checks allows the Ball to move properly in a confined environment.

Evaluation

Currently, the game can be launched and it has the proper look and feel of a Brick Breaker game. Moreover, the additional Depth element can be controlled easily and feels intuitive. The ball bounces from the blocks, tray and the walls, and properly destroys the blocks on contact. The orbital mouse control allows the user to change their viewpoint whenever they desire. This way, the fact that the ball can move in the Z-axis can be countered, and the 3-dimensional movements can be detected by rotating the view.

Aside from the successes of the project, there exists a problem with the collision detection, and the ball can sometimes miss the walls, going through them, and move outside the

scene, flying forever. This is probably due to the step size. The ball probably moves in such a way that its boundaries never actually hit the boundaries of the walls, jumping without causing a collision. One of the future works of this game would be fixing this collision problem, and a possible fix is to check next and previous steps to be sure about where the ball was and where it will be.

Another future addition that is crucial to a Brick Breaker game is that different contact points on the user's tray should reflect the ball differently. Currently, the ball reflects directly off the horizontal surface, reversing the ball's motion in the Y direction. However, this is not always the case in regular Brick Breaker. Hitting the ball from the left side of the tray should reflect the ball more to the left, and the same for the right. In addition, adding this functionality would mean that hitting the ball from the bottom or top parts of the tray would also affect the ball's motion in the Z direction, since depth is also an element in this 3D game.

Conclusion

Brick Breaker is a very old game that has been implemented in 2D, and this project is an attempt to visualize it in 3D. It further aims to add a different perspective to the gameplay and its mechanics. By implementing it in a 3D world and adding a domain specific element that is the Z direction, the project has extended upon the original game and at the same time shown the Brick Breaker environment rendered with the Computer Graphics tools learned in the course, using Java3D's methods.

The project is generally working correctly considering the overall look and feel. It functions as how a Brick Breaker game should function: The tray behaves correctly according to the given user commands, which are the mouse movements. As the user moves the mouse, the X-coordinates of the tray are positioned according to the X-coordinates of the mouse and the Z-coordinates of the tray are positioned according to the Y-coordinates of the mouse. The blocks are also destroyed on contact and therefore this functionality is also present. The blocks may be equipped with a logic in the future that causes them to generate several bonuses for the user when they are destroyed. In addition, the ball moves and bounces off surfaces in a predictable fashion. In case the bugs are fixed, and the different bouncing positions on the tray are

implemented, then this game can become a fun and playable game that is fresh and somewhat different than the classic.

The project has been implemented with the flexibility provided by Java3D, and the knowledge granted by the Computer Graphics course. Using the information provided in this class, additional visual improvements have been made to create a visually attractive game. Visual effects such as lighting, material, transparency, and borders have made the game better looking and thus more compelling in terms of the execution of Computer Graphics.