Custom Pool 3D

By

Tom Wesley

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

May 2002

Acknowledgements

I would like to thank all those at the Morfit 3D Website who helped me get accustomed to using the Morfit 3D engine, especially Chandan”Chand” Pawaskar and “Bucky” Bucanan whose replies to my posts on the Morfit 3D engine Website helped me to overcome some of the major obstacles I had when developing this game.

Table of Contents

List of Figures

Abstract

Custom Pool 3D is a 3D billiard game that allows the user to create his/her own table designs. The game idea originated with an old 8 bit Nintendo game called *Lunar Pool*. This game used simple physics and had 99 different tables on which to play. I decided to develop one that would not only have a multitude of boards but that also would be played in a 3D environment and give the user the ability to create his/her own tables. *Custom Pool 3D* uses a simple board editor: the user can create a board in about any shape. Once the user's design is complete the table is compiled, rendered, and played in 3D. *Custom Pool 3D* was created using Visual Basic and the Morfit 3D engine.

Custom Pool 3D

1. Statement of Problem

Custom Pool 3D fills a void in the video game market that has been empty for a while. There have been some pool games created lately but none that do as much as this one. I have searched for possible competitors and found that the Lycos search engine only lists three pool games and Yahoo doesn't list any. When compared to the number of first person shooting games and role playing games, the market for billiard games of any type is wide open. The best of the existing games is "Real Pool 3D" created by Digital Fusion but this game is more of a simulator. It's main goal is to seem as real as possible(hence the name). This game was part of my inspiration to make Custom Pool 3D. The game has a couple boards that aren't rectangular but they are only 1 player levels and they all have a certain goal or objective to complete, no free play. There was Lunar Pool which was a 2D overhead view billiard game that was nothing but different shaped boards. This game was created for the original 8 bit Nintendo system so it is outdated. My Custom Pool 3D will merge the two games together and then go a step farther with the added options. I believe that the uniqueness of the game will make it a marketable product. To further support my theory that this game will make money I have interviewed many "hardcore gamers" that I know and have received nothing but enthusiasm about the concept of the game. I plan to distribute my game through the manufacturer of the 3D engine I will use. Morfit.com offers a marketing tool for games made with their engine. For 50% of the profits generated by the game the Website will take care of all the marketing and distribution of my product. The only stipulation is that they only distribute products that they feel will create enough profit to justify their work. This does not have to be the only way to market my product but does seem like the most feasible means right now. There is nothing in the agreement with Morfit that will prevent me from pursuing other forms of distribution in conjunction with theirs.

2. Solution

2.1 User Profile

Users with enough understanding on how to operate a computer in the most general sense will have enough IT experience to operate my program.   Some experience playing 3D games might make their initial exposure to the game a little easier but all the help menus and directions will be geared for even the most computer illiterate user. The only requirement I have for a user is the desire to waste valuable time in front of a computer playing video games.

2.2 Design Protocols

2.2.1   Organizational Scheme

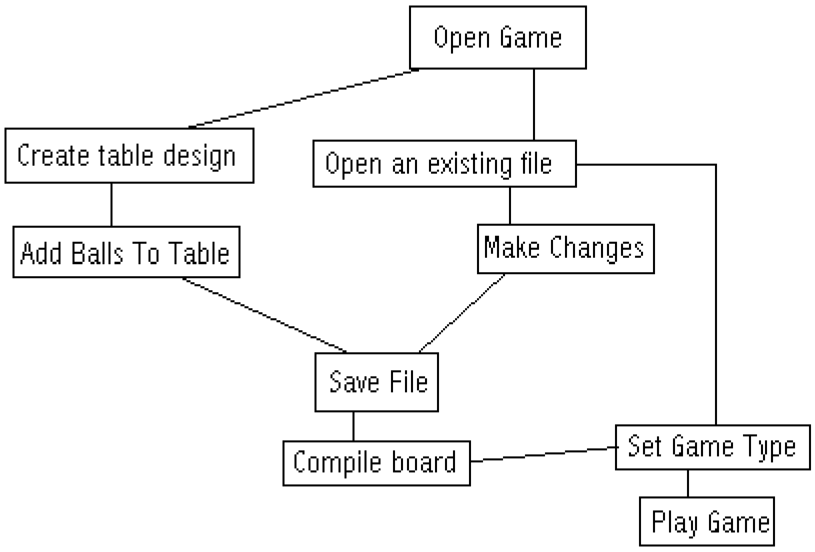

Figure 1. Organizational Scheme

2.2.2 Interface

There is one application that will give the user access to the board editor and the game itself. The game opens into the board editor where the user will either create a new board or load an existing board. If the user creates a new board or changes an existing one they will have to recompile the board before they can play it. If they choose to play the game with the board as it is then all they have to do is click the play button and the 3D board will be loaded and the game will commence from there

2.2.3 GUI Standards

Only the board editor will need icons and the like. In the board editor above the board layout picture box is a toolbar with all the items the user can pick from when creating the board.

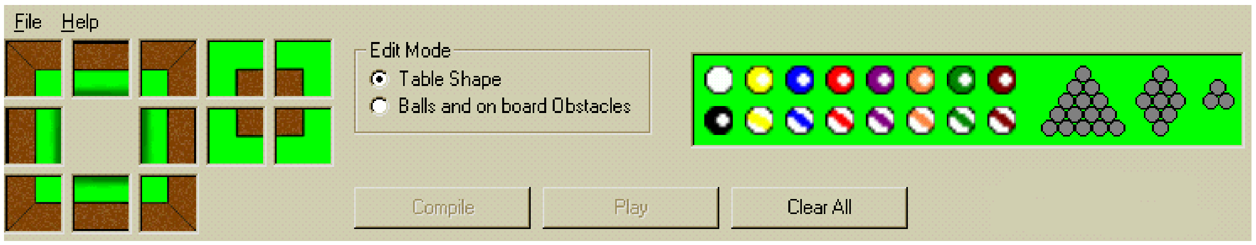

Figure 2. GUI Interface

Here are the buttons for the GUI interface. The boxes on the left hold the types of bumpers and on the right are the balls and racks. Once the board is created and checked for integrity of design the balls and racks icons will become enabled and the user can finish the design of the board. For example if the user wanted to place a piece of a bumper that ran horizontal and faced down they would click on the picture box that looked like this: 🟫.

Here is an example layout of a completed board.

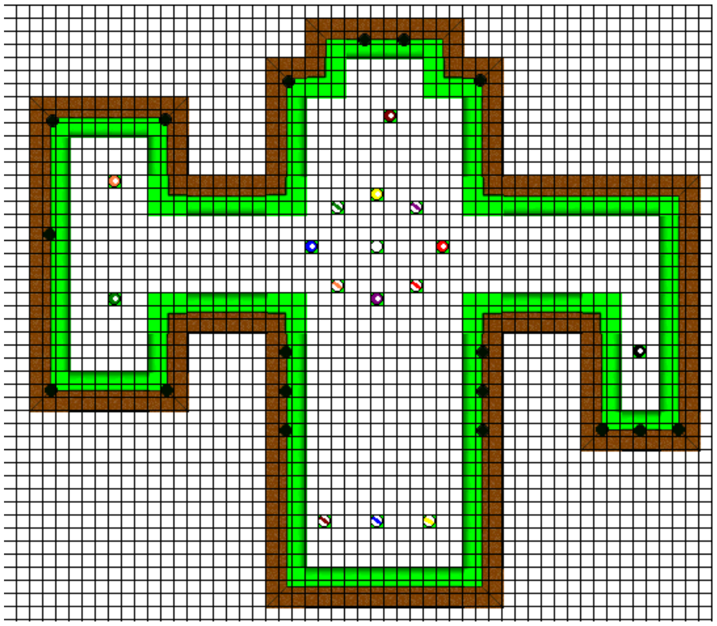
Figure 3. Sample Board Editor Table Layout

The game itself will be played completely without the use of icons. The user will click and drag and move the mouse around the form to interact with the 3D environment which will be painted onto the form. When the user is trying to line up a shot he/she holds down the right mouse button and drags the mouse around the ball. When the user is satisfied with the angle of the shot he/she holds the control key and left mouse buttons down to move the stick in order to hit the ball.

2.2.4 Color Scheme

Since the interfaces are composed of Window's forms there is no color scheme that will be used so that the forms will all reflect the user's current desktop settings.

3 Deliverables

3.1 Physics of the game.

<u>Collisions based on 2d environment</u>. Although the game will be in a 3D environment there will be no third dimension to the game play. The user will be able to move around the

environment in a 3D fashion but the actual physics of the game and the way the different pieces of the game interact will not allow the pieces to move in 3D. Basically there will be no z axis movement.

Limited number of physical variables. Ball rotation will not be simulated. The balls will collide with each other, the bumpers, and various table obstacles similar to the way a puck moves on an air hockey table. The game will make use of a balls mass when calculating collisions. There will also be a coefficient of restitution variable that can be adjusted for the bumpers. This variable determines the elasticity of the bumper and therefore will create bumpers that have more and less bounce. The table itself will also have a friction variable that can be changed to make the balls slow down at various rates.

3.2 Board editor options and abilities

The board editor will be a 2D Visual Basic application that will give the user the ability to create a custom shaped board with different physical attributes and ball layouts.

Create custom boards. The board editor's main purpose is to allow the creation of differently shaped tables to play on.

Saving. The game allows the player to save the boards created in an editor format that can be read by the board editor and a format that will be used by the game itself to create the board in 3D (this file is created when the table is compiled).

Ball placement. Not only will the user be able to create alternate shapes for the board but the user will also be able to place the balls on the table anywhere they want. They will have the ability to place the balls on the table in a specific rack alignment such as an eight ball rack or nine ball rack or place the balls individually on the board.

Physical variable adjustments. All physics variables will be able to be adjusted here. The mass of each individual ball and the coefficient of restitution for each bumper. The frictional force created by the felt of the table can also be adjusted here.

3.3 Game Play

The game includes rules for game play based on popular billiard games. These games are eight ball, nine ball, three ball, and cutthroat. There is also support for user defined games.

Eight ball. All 15 balls are arranged in the traditional rack. Each player is designated stripes or solids. The object of the game is to pocket all your balls and then sink the eight ball before your opponent does. If a player pockets the eight ball before clearing all of their balls that player looses.

Nine ball. The balls are arranged in a diamond shape using only the balls numbered 1-9. The object of this game is to sink the balls in numerical order up to the nine ball. You may win without clearing all the balls first if you hit the current object ball first into a combo that hits the nine ball in or if some ball hits the nine ball in after hitting the object ball.

Three ball. Three balls are arranged in a triangle shape. Each player takes a turn trying to shoot all the balls into the pockets. When everyone has had a chance to hit all the balls in, the player who accomplished this with the least number of hits wins. This is mostly a gambling game but is still fun.

Cutthroat. game for three players in which a set of five balls is designated to each player. The object of the game is to knock the ten balls, designated to your opponents, into the pockets before your five balls are pocketed. This game uses the traditional eight ball rack.

User defined games. These are games where the user decides not to use conventional rules and ball arrangements. In these games the user will be able to apply existing rules to their game. For example the user could scatter the balls all around the table then apply nine ball rules to the game and then set the final ball to which ever ball they would like. In this example the user could use all 15 balls and set the final ball to the 15 ball. So in essence the players will have to hit all the balls into the pockets in numerical order up to and including the 15 ball but there would be no rack to break from.

4. Proof of Design

Custom Pool 3D was a large and ambitious project. There were many hurdles that were overcome, most of which were unseen at the planning stage of the project. Most of my deliverables were met, but there were a few that did not get fulfilled or were only partially completed.

4.1 Game Physics Deliverables

Collision handling turned out to be far more complicated than I originally thought and was the cause of most of the delays in meeting my schedule. For the most part the collision handling works, but there are a few cases where things go awry. These are mostly when the cue ball is hit exceedingly hard. Because of the way that the collisions are detected, the balls occasionally will stop when they aren't supposed to. I could have limited the amount of force that the user could hit the ball with but this would have made shots that required a lot of distance impossible. As stated earlier the game treats the balls like pucks on an air hockey table and the only movement along the Z axis is when the ball goes into a pocket. This is done completely manually and not handled by an enforcement of the rules of physics.

The biggest drawback of the collision handling difficulties is that racks were for the most part useless. When you attempted to break the balls from a rack most if not all of the balls remained stationary upon impact.

The use of physical variables such as mass and the bounce factor of the bumpers was not implemented in the game portion of the project. The mass was left out of the collision handling because of time constraints. The cooefficient of restitution was left out because of the way the engine handles dynamic and static objects. At runtime there is no way to differenciate one static object from another. The big problem with this is that the more dynamic objects in a world the slower the frame rate is. Because of this I decided to keep all the bumpers static. Since the bumpers were all static and there is no way to tell which bumper piece I have hit there was no way to enforce the individual coefficients of restitution.

4.2 Board Editor Deliverables

The board editor was a complete success. I have fully implemented the ability to create pool table of any shape the user wishes. Along with crating the boards the player can also save the boards and open them at any point in the program. The only limitation to this is that the board must have a valid design before it can be saved. A valid design simply means that a complete barrier is created, meaning that there are no holes in the outline of the board. Balls can be placed on the table individually or in a rack of their choosing. The racks that can be placed are completely customizable but also offer a default setup.

Although the physical variables were not implemented in the game, they were implemented in the board editor. This was because the board editor had to be completed before the game portion of the project was, and the limitations of the 3D engine was not yet fully understood.

4.3 Game Play Deliverables

I had originally included four predefined types of game and a user defined game. I was unable, due to time, to implement the three ball and cutthroat games. The eight ball and nine ball games have been almost completely implemented. The only thing that was left out was the ability to place the ball anywhere during game play. Right now if a player scratchs the cue ball simply goes back to it's original location when the board was loaded. All other rules were enforced.

The user defined game was implemented successfully. The first example is that the game can be played with no rules at all which is basically a user defined game. Along with no rules the game can also use any number of balls and play nine ball. The user must specify which ball is the final ball and all the rest must be hit in numerical order before he/she is allowed to hit the final ball in, unless they hit the object ball first and as a result the final ball is sunk. If the user specifies a ball that is not the ball of highest numerical value then all the balls with a value higher than the final ball are just obstacles that can be used to continue a turn if sunk after hitting the object ball first. Eight ball rules can also be applied to a custom game by not placing all the balls on the table. The rules used for eight ball dictate that player onest be solids and player two is stripes and that the eight ball is the final ball. Those are the only limiting rules when creating a custom eight ball game. With these rules the user could place all the solids on the table and only some of the stripes giving player 2 an advantage, or you could only place three balls of each type on the board and play that way.

5. Conclusions and Recommendations

In conclusion the game was a success. I had set out to create a fully functional pool game and for the most part I've hit my mark. The game may not be the prettiest 3D game out there but it is a 3D game and I never made any promises on the graphics. Which is a good thing because on the machine this was created on, the implementation of more detailed bitmaps and the use of dynamic lighting would have greatly depreciated my frame rate which is already low. This was one of the major drawbacks I found with the engine. I attempted to optimize the engine's variables but nothing seemed to make much of a difference. I am eager to find out how it looks on a newer more robust computer.

Looking back on the entire project I don't think I would have picked a different engine even with it's limitations. The Morfit 3D engine allowed me to develop the application in the language of my choice which had more drastic effects on my timeline than the engine did on game play. If I had to develop this game in C++ it would not have turned out quite as well as it did.

There were a few difficulties with the editor. The first was how to determine if the user had created a complete board. In order to do this I called on some of my C++ programming skills and basically implemented a link list. Each bumper had a left and right property that linked it to another piece. When the user saves the design I traverse this linked list and make sure that I get back to the beginning piece. Along the way each piece is flagged as being accounted for. I then check to make sure that each piece that is on the board has been counted. The other problem that arose was how to fill in the area inside the board so that the balls didn't just float in mid air. This turned out to be a major accomplishment in and of itself. The filling algorithm that was written for this is some of the most complex code that I have written.

The game itself seemed to progress well, at least until I got to the physics and collision handling. In order to keep the balls from colliding with each other more than once per rendering cycle (i.e. finding that ball one hit ball two then later finding that ball two hit ball one) and to speed up the main loop I had to implement a permutation list that held all the possible combinations of two balls. This cut down the number of collision checks to a fraction of the original number which in effect increased my frame rate a bit.

I believe that the main reason why my frame rate is so low on the computer it was developed is that the number of polygons is excessively large because the table is created from many different components. Given more time I think that this could be handled by dynamically stretching different pieces instead of using multiple small pieces. This would be greatly increased if the Morfit Engine made use of cut brushes. Cut brushes allow you to make a 3D object that cuts away from other objects. Instead of creating a bumper with a hole in it I could have dynamically placed these cut brushes where the holes should have been. I think that this would cut down on the polygon count for the holed bumpers and also allowed for more flexibility in the placement of the holes on the bumper.

The last thing that should be addressed is the collision handling itself. This turned out to be far more difficult that originally thought. For one there are multiple different ways to implement it. The most difficult of which would be real time. In this method you move the objects depending on the amount of time that has elapsed. This is a more efficient model but is also more involved. One of the main benefits of doing this is that the game will run at the same speed regardless of the speed of the computer it is run on. I had to opt for another version due to the time constraints of the project. I used a method that simply moves the balls according to their speed during each rendering cycle. This will cause the balls to appear to move faster on faster computers because the resources available to a more powerful computer will increase the frame rate and process the information in the loop quicker. The other issue with the collision handling is that there was far more physics involved than expected. I was planning on having just a few equations that needed implementation when in fact there were many more than a few. There is also different ways to handle moving the balls. At first I used the physics supplied by the engine. I found this to be insufficient for my project. I then created my own physics handling algorithm which ended up being too complicated and too slow. In the end I used a hybrid of both methods. I used the physics in the engine to move the balls around the table then used my own code to handle the collisions with the different objects. The switching between the three methods took an excessive amount of time and is the main reason why some of my deliverables were not met.

I believe that a stronger knowledge of available 3D engines and methods used to create and design efficient 3D worlds and objects would have made a large difference in the final product, but then again that was the idea behind taking this project on. I have definitely learned a lot about 3D games and will probably continue development on this and other projects as a hobby. About halfway through the development of the 3D portion of the program I was beginning to hope that I had developed a 3D shooting game or something similar because the physics and handling of moving objects would have been far less difficult. I would advise anyone thinking of taking on a similar project as mine to have extensive knowledge of the 3D software they are using and have some experience in implementing the laws of physics. It's one thing to understand them, and another to tell a computer to follow them.

Appendix A.
Time Line

September	1st 21st 26th	Create 3D versions of pieces in board editor Create “Save as game file” functionality to board editor Begin functionality to assemble 3D pieces in game
October		Finish assemble in 3D functionality Create functionality to play a game of 8 ball
November	1st 16th 21st	Create functionality to play a game of 9 ball Create Beta Version Install program and test Create Help
December	1st 5th 5th	Create Rules and Regulations for Beta Test First Beta Test Session Make changes as problems develop
January	1st 15th 29th	Create functionality to play a game of 3 ball. Create functionality to play a custom game. Update Help

February	1st 6th 6th	Update Install program and test. Second Beta Test Session begins Make changes as they develop
March	 16th	Second Beta Test Session Ends on 15th. Make Changes as problems develop. Finish all changes from beta Test.
April	1st 1st 1st	Create Finished product. Improve Graphics if time available Add extra’s not in deliverables if time available
May	1st 1st	Final testing and debugging. Development of Presentation and Final Paper
June	6th	Presentation

Appendix B.
Budget

Item	**Cost**
Morfit 3D Engine(Student License)	$ 49.00
MilkShape 3D	25.00
Visual Basic 6 (Learning Edition)	99.00
Estimated Value of Computer Used For Development	300.00
Adobe Photoshop	135.95
Total	**$608.95**

Appendix C.
Software

There were three main programs used to create Custom Pool 3D, the Morfit 3D engine, Visual Basic 6 (Learning Edition), and Milkshape 3D. The Morifit engine was used for all the game rendering. It is the backbone of the project, without it there would be no 3D in Custom Pool 3D. This program allowed me to render the table to a VB form and to manipulate all the objects in the 3D environment. Visual Basic 6 is what I used to implement the game. With VB I was able to interact with the engine and make API calls to the engine's functions and subroutines which allowed me to take user input and apply it to the 3D environment. I also used VB as the only tool to implement the board editor. Milkshape 3D was used to create all of the 3D models used in the game. I also used Milkshape to skin some of these objects. Milkshape was not a necessity but I found that the world editor supplied with the Morfit 3D engine was too cumbersome and confusing to use. By creating all my models with Milkshape my development time was cut significantly, not to mention my mental well being was preserved.

Appendix D.
Hardware

This project was developed on a Gateway Pentium II, 450 mhz computer with 10 gbs of hard drive space and 256 mbs of RAM.

Bibliography

1. "3D State/Morfit Website". http://www.3dstate.com/Developers/developers.htm. 2002.

2. Cutnell. *Physics*. Canada: Von Hoffmann Press, 1998.

3. Deitel, H. M.. *C++ How to Program*. New Jersy: Prentice-Hall, 1994.

4. Edwards. *Single-Variable Calculus With Analytic Geometry*. New Jersey: Prentice-Hall, 1994.

5. Lander, Jeff. "Physics on the back of a cocktail napkin". http://www.gamasutra.com/features/20000516/lander_pfv.htm. May 16, 2000.

6. Shahar, Tzachi. *3D Web Site Programming with Northdragon.* India: Northdragon, 2001.

7. Spenik. *Visual Basic 6 Interactive Course*. Corte Madera, CA: Waite Group Press, 1998.