

**3D COMPUTER SIMULATION OF A BOUNCING LIGHTWEIGHT  
SPHERICAL BODY**

**A Research Paper  
Presented to  
The Faculty of Philippine Science High School Western Visayas  
Bito-on, Jaro, Iloilo City**

**In Partial Fulfillment  
Of the Requirement for  
SCIENCE RESEARCH 2**

**by**

**Paulo Miguel B. Lacro-o**

**Fourth Year Graviton**

**March 2009**

## Table of Contents

Approval Sheet  
Acknowledgements  
Abstract

### CHAPTER

#### I. INTRODUCTION

A. Background of the Study	1
B. Statement of the Problem	2
C. Objectives of the Study	3
D. Significance of the Study	3
E. Definition of Terms	4

#### II. REVIEW OF RELATED LITERATURE

A. Computer Simulations	6
B. Table Tennis	7
B.1 Spin	7
B.2 The Effects of Spin	8
B.3 The Ball, the Table, and the Paddle	11
C. Other Concepts	13
C.1 Kinematics	13
C.2 Gravity	14
C.3 Newton's Laws	15

#### III. METHODOLOGY

A. Overview of the Methodology	16
B. Methods	17
B.1 Gathering of Materials and Physics Concepts	17
B.2 Development of the Physics Engine	18
B.3 Development of Graphics Engine	21
B.4 Development of the Shell Program	21
B.5 Testing of the Final Program	22

#### IV. RESULTS AND DISCUSSION

A. Results	23
B. Discussion	25

#### V. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

A. Summary	28
B. Conclusions	28
C. Recommendations	28


#### LITERATURE CITED

## APPROVAL SHEET

This Research Paper Hereto Entitled:

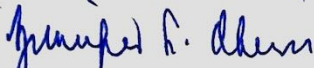
“3D Computer Simulation of a Bouncing Lightweight Spherical Body”

Prepared and submitted by Paulo Miguel B. Lacdo-o in partial fulfillment of the requirements in Science Research 2 has been approved and is recommended for acceptance and approval.




JOSEPH SIMON V. MADRIÑAN  
Science Research 2 Adviser

Approved by the committee in oral examination with a grade of PASSED on March 2009.




ZENNIFER L. OBERIO  
Member



ARIS C. LARRODER  
Member



MIALO C. LACADEN  
Member




EDWARD C. ALBARACIN  
Member



OLIVER J. FUENTESPINA  
Member

Accepted in partial fulfillment of the requirements in Science Research 2



JOSETTE T. BIYO, Ph. D.  
Director III – PSHSWVC

## ACKNOWLEDGEMENTS

First of all, I would like to thank my parents for always reminding me of what to do, always watching over me, caring and loving me. Thank you for always being there to support me and help me in everything.

I would also like to express my gratitude to the research teachers who taught us what we need to know to be able to finish our paper. Thank you for sharing your knowledge with us. I impart my sincere gratitude to Sir Joseph Madriñan for being patient and understanding, and for being my research adviser.

Thanks also to my classmates and friends for being always there no matter what, and no matter how terrible the situation was. They were always there to tell me and help me.

Thanks to my class adviser, Ma'am Evaline Gerochi, for always extending her arms to help each and every one of us in the class. Thank you Ma'am Val for telling me to walk the talk. Also thank you to all the teachers who advised me on what to do on hard times.

Finally, thanks to Greeny because somehow, she acted as my conscience.



Lacdo-o, Paulo Miguel B. "3D Computer Simulation of a Bouncing Lightweight Spherical Body." Unpublished Research Paper. Philippine Science High School Western Visayas Campus. Brgy. Bito-on, Jaro, Iloilo City. April 2009.

## ABSTRACT

A computer simulation is an imitation of many natural systems in an attempt to model these systems and gain insight in their operation. Simulations are often used in the training of civilian and military personnel. Physics computer simulations are used to model many real life situations or activities like playing a game of billiard, stacking of objects called rigid bodies, and the bouncing of a ball.

There are simulations that focus in using new algorithms to simulate physics environments realistically. Some tried to create a basic framework for further development. Others are simply for leisure.

This study aimed to create a computer program that can simulate the bouncing of a lightweight spherical body following the laws of physics that governs its motion. The program was coded using C++ as its programming language. DirectX was used for the graphics, and Windows was the platform used.

The study produced a computer program that can simulate the effects of aerodynamics and gravity on a bouncing ball.

## CHAPTER I

### INTRODUCTION

#### A. Background of the Study

A simulation is an imitation of a real thing, state of affairs, or process. Computer simulation has become a useful part of modeling many natural systems in physics, chemistry, and biology, and human systems in economics, and social science as well as in engineering to gain insight the operation of the said systems. Simulations are also often used in the training of civilian and military personnel (Wikipedia).

There are plenty of simulations that deal with physics. A study entitled “Non-convex Rigid Bodies with Stacking” aimed to construct plausible motion instead of predictive motion. It used triangulated surfaces for geometric representation. A new algorithm for collision detection and modeling was proposed, which fixed problems with the previous algorithms. It used contact points, contact graphs, shock propagation, and included friction (Guendelman and others ).

A school project used the ideas and algorithms from the research paper by Guendelman and others. Its goal was to create a solid framework for simulations of rigid body dynamics. The project used C++ as the programming language and OpenGL for rendering. The software supported four primitives: spheres, immovable planes, cubes and general boxes. Only the spheres and immovable planes were allowed for collisions. Simple Euler integration was used for solving ordinary differential equations (ODE) (Lewis). The collision detection was made with the use of bounding boxes, wherein when

two bounding boxes overlap, collision is detected. The advantage of this is simpler computations can be made. The disadvantage of this is that the project was using spheres. Bounding boxes for spheres have spaces on the corners that may detect collision even though the spheres have not yet collided.

In another study, Hsieh and Chang also made a rigid body simulation. Their goal was to create a realistic simulation of rigid body dynamics, which was a billiard game. The ideas and algorithms were based on SIGGRAPH97' Course Notes Unconstrained Rigid Body Simulation and Constrained Rigid Body Simulation, both by Baraff and Witkin. They used quaternion to represent rotation and some other attributes. Euler integration was also used in solving ODEs. They used OpenGL for rendering graphics and Fast Light Toolkit for handling Windows and user inputs.

This study aims to develop a computer software program that can simulate the path of a spinning lightweight spherical body (table tennis ball) and give realistic physics-based results which will be rendered in 3D graphics.

## **B. Statement of the Problem**

This study aims to design computer software that can simulate the path of a spinning lightweight spherical body, a table tennis ball, and give plausible physics-based results.



### **C. Objectives of the Study**

This study has three objectives:

1. To design a physics engine that can compute all the physics computations of the software using C++.
2. To design a graphics engine that can handle the 3D transformations and translations, and rendering of the software using C++ and DirectX. The engine will also have an added capability of handling inputs.
3. To design a shell program that can integrate the physics and graphics engines, and handle all required processes of the software. The shell program will also handle Windows processes involved with the software, and will be written using C++.

### **D. Significance of the Study**

The software was designed to use DirectX for rendering graphics and handling user inputs. The purpose of such is to make the design of the software focused on the physics computations instead of handling graphics and inputs.

The software will help hobbyists and students see the effects of spin on a ball, common on most ball sports game like table tennis, lawn tennis, football, and baseball, without actually being in the game or playing it.

The software can also be used as a basis for making computer games that involves spins.

## E. Definition of Terms

**C++** - is a general-purpose programming language. It is a statically typed, free-form, multi-paradigm, usually compiled language supporting procedural programming, data abstraction, object-oriented programming, and generic programming (<http://en.wikipedia.org/wiki/C%2B%2B%2B>).

In this study, C++ is the programming language that will be used in writing the code of the program.

**Computer software** – is a general term used to describe a collection of computer programs, procedures and documentation that perform some task on a computer system (<http://en.wikipedia.org/wiki/software>).

**DirectX** – is a collection of application programming interfaces for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms (<http://en.wikipedia.org/wiki/DirectX>).

In this study, DirectX is the API that will be used in writing the graphics engine.

**Physics** – is the science of matter and energy and of interactions between the two, grouped in traditional fields such as acoustics, optics, mechanics, thermodynamics, and electromagnetism, as well as in modern extensions including atomic and nuclear physics, cryogenics, solid-state physics, particle physics, and plasma physics (<http://www.thefreedictionary.com/physics>).



**Physics engine** - is a computer program that simulates Newtonian physics models, using variables such as mass, velocity, friction and wind resistance. It can simulate and predict effects under different conditions that would approximate what happens in real life or in fantasy world ([http://en.wikipedia.org/wiki/Physics\\_engine](http://en.wikipedia.org/wiki/Physics_engine)).

In this study, the physics engine is the part of the software responsible for solving computations of the program.

**Plausible** – means superficially fair, reasonable, or a valuable but often specious (<http://www.merriam-webster.com/dictionary/plausible>).

In this study, plausible is to be close to reality.

**Rendering** – is the process of generating an image from a model, by means of computer programs ([http://en.wikipedia.org/wiki/Rendering\\_\(computer\\_graphics\)](http://en.wikipedia.org/wiki/Rendering_(computer_graphics))).

In this study, rendering is to display the image on the computer screen.

**Table tennis** - is a sport in which two or four players hit a lightweight, hollow ball back and forth to each other with paddles or rackets. The game takes place on a hard table divided by a net. Players must allow a ball played towards them only once bounce on their side of the table and must return so that it bounces on the opponent's side ([http://en.wikipedia.org/wiki/Table\\_tennis](http://en.wikipedia.org/wiki/Table_tennis)).

## CHAPTER II

### REVIEW OF RELATED LITERATURE

#### A. Computer Simulations

A computer simulation is an attempt to model a real-life or hypothetical situation on a computer. In that way, the situation can be studied to see the system works. By changing variables, predictions may be made about the behavior of the system (Wikipedia).

Computer simulations have become a useful part of modeling many natural systems in physics, chemistry and biology, and human systems in economics and social science and also engineering to be able to gain insight into the operation of those systems (Wikipedia).

Rigid Body Simulation was a school project by Darren Lewis. The ideas and algorithms were based on the paper Non-convex Rigid Bodies with Stacking by Eran Guendelman, Robert Bridson and Ronald Fedwik. He attempted to create a solid framework for simulations of rigid body dynamics. The programming language he used was C++. He used OpenGL for rendering the graphics. The software supported four primitives: spheres, immovable planes, cubes and general boxes. He used simple Euler integration for solving ordinary differential equations (ODE).

The paper by Eran Guendelman and others mentioned on the paragraph above had a goal of constructing plausible motion instead of predictive motion. They proposed a new algorithm for collision detection and modeling. The new algorithm fixed some problems with the previous algorithms. They used triangulated surfaces for geometric representation.

Chen-Nan Hiseh and Chia-Ming Chang also made a rigid body simulation. Their goal was to create a realistic simulation of rigid body dynamics which was a billiard game. The ideas and algorithms were based on SIGGRAPH97' Course Notes

*Unconstrained Rigid Body Dynamics* and *Constrained Rigid Body Dynamics*, both by Baraff and Witkin. Euler integration was used in solving ODEs. They used OpenGL for rendering graphics and Fast Light Toolkit (fltk) for handling Windows and user inputs.

## B. Table Tennis

Table tennis is a sport in which two or four players hit a lightweight ball back and forth to each other with rackets. The game takes place on a table divided by a net. Players must allow a ball played towards them bounce only once on their side of the table and must return it so that it bounces on the opponent's side. Play is fast and demands quick reactions. A skilled player can impart spin to the ball, which makes it bounce and its reaction on the opponent's racket difficult to predict or return with confidence (Wikipedia).

### B.1 Spin

A ball that is spinning is also easier to return than a ball that is not spinning because a ball that is spinning has stability at range. A simple explanation of this is a rifle. There are *lands* down the inside of the barrel. These *lands* are grooves that twist in one direction, causing the bullet to spin. Without the *lands*, the bullet would stray after about 50 meters (Letts).

A spinning ball is also harder to receive. A ball imparted with a spin has a curved path out of the ordinary parabolic path driven by gravity. There are three types of spin and each has a descriptive path of its own.

#### B.1.1 Topspin

A ball that is topspun has an axis of rotation along the x-axis. It rotates clockwise when viewed from the positive x-axis<sup>a</sup>. It has the tendency to drop faster than the acceleration due to gravity. It also tends to bounce off the table flatter and faster (Letts).



### **B.1.2 Underspin**

Sometimes also known as bottomspin, a ball with this spin has its axis of rotation along the x-axis like a topspun ball. It rotates counter-clockwise when viewed from the positive x-axis. It has the tendency to drop slower than the acceleration due to gravity unlike a topspun ball. It also seems to float on to the other side, and tends to bounce off the table higher (Letts).

### **B.1.3 Sidespin**

A ball that is sidespun has its axis of rotation along the y-axis. It tends to curl either to the left or to the right depending on the direction of its rotation. When viewed from the positive y-axis, a ball that is spinning clockwise has a path curving to the right, while a ball spinning counter-clockwise has a path curving to the left (Letts).

## **B.2 The Effects of Spin**

### **B.2.1 Moving through air**

The flight of a ball through air, like the path of any object through a fluid, depends upon the properties of both the object and the fluid. These specific properties influence the path that the ball takes. In particular, the path of a ball in flight is strongly affected by air resistance. The forces on a ball from air resistance are typically of the same magnitude as gravity, so they have quite noticeable effect (Berger 2000).

### **B.2.2 Bernoulli's Principle**

One of the most fundamental concepts in all fluid dynamics is the streamline. Streamlines are lines tangential to the velocity vector at each point in a fluid at a single moment in time. For steady flow, a streamline actually traces the actual path of an individual fluid particle. By drawing streamlines, one can accurately visualize the flow around a ball (Berger 2000).

While an individual streamline indicates the direction of the fluid flow, the density of all streamlines indicates its magnitude. The flow is faster over the top and bottom of the ball than it is further away. The velocity and pressure of a fluid at two points along a streamline are related such that

$$p_1 / \rho + gy_1 + \frac{1}{2} v_1^2 = p_2 / \rho + gy_2 + \frac{1}{2} v_2^2$$

Here,  $\rho$  is the density of the fluid,  $p$  is its pressure,  $y$  is its height, and  $v$  is its velocity. This equation is called Bernoulli's Equation (Berger 2000).

### B.2.3 Viscosity

Viscosity is a measure of the friction among adjacent layers of fluid, parallel to the fluid flow. It manifests itself in the fluid's resistance to change shape.

The relative effects of viscosity for a particular case are represented by a dimensionless constant called the Reynolds' Number. The Reynolds' Number is a value assigned to a dynamical system that describes the character of its fluid flow. It is proportional to the density of a fluid, and the size of an object, and inversely proportional to the viscosity of the fluid:

$$R = \rho v d / \mu$$

A small value of the Reynolds' Number describes a system in which viscous forces dominate, and a large value indicates that inertial forces dominate (Berger 2000).

### B.2.4 Boundary Layer

Compared to most fluids, the viscosity of air at atmospheric pressure is very small. The air far away from moving baseball is essentially unaffected by frictional forces. However, air molecules close to the baseball may be greatly affected. A thin layer of fluid develops very close to the baseball may be greatly affected. A thin layer fluid develops very close to the sphere in which particles experience a shearing stress. This layer of fluid is called a *boundary layer*, credited to Ludwig Prandtl (Berger 2000).



The boundary layer appears for systems described by a large Reynolds' Number. According to Prandtl, "Through fluid with only slight friction behaves just like a frictionless fluid at places where there is no boundary, a thin 'boundary layer' is formed at the walls as a result of friction". The value of velocity within the boundary layer varies from that of frictionless motion (away from the sphere) to that which adheres to the boundary. The frictional forces away from the boundary are negligible, but close to the boundary, they are on the same order of magnitude as the inertia of the fluid. No matter how small the viscosity of the fluid is, any body moving through it drags along a thin layer of fluid (Berger 2000).

One factor that may affect the boundary layer is the turbulence that is induced in the surrounding air. Turbulence can penetrate a uniform boundary layer and greatly alter the character of the airflow past a baseball. For turbulent flow differs greatly from the smooth, laminar airflow is typical for lower velocities (Berger 2000).

### B.2.5 Resistance

The phenomenon of boundary layer separation is what leads to the air resistance experienced by a ball. The chaotic flow in wake region leads the pressure on the back of the ball to be much lower than the pressure on the front of the ball. The kinetic energy lost by air molecules in the boundary layer of the ball bound up in the eddies in its wake. It is not reconverted to potential energy. The existence of the wake creates a drag force on the ball (Berger 2000).

The magnitude of this drag force is proportional to the difference in pressure from front to back times the area over which it acts. Bernoulli's principle provides a model to deal with the pressure difference. At the front of the ball, fluid particles essentially dammed up, with zero velocity. Behind the ball, they ultimately leave with velocity  $v$ . By Bernoulli's equation, the difference in pressure is

$$p_1 - p_2 = \frac{1}{2} \rho v^2$$

Thus, if  $A$  is the area of the ball, the total drag force is

$$F_d = \frac{1}{2} \rho a v^2 \times C_d$$

$C_d$  is the drag coefficient, whose value is a function of the Reynolds' number.

### **B.3 The Ball, the Table, and the Paddle**

In table tennis, not only the path of the ball through the air is taken into account but also the bounce of the ball on the table and the player's paddle. Although spin affects the way the ball bounces on both table and paddle, the effects of spin can not be observed without the other factors involved in collisions.

#### **B.3.1 Momentum and Collisions.**

The momentum of a body is defined as the product of its mass and its velocity. Momentum is usually represented by the symbol  $p$ .

$$p = mv$$

A force is required to change the momentum is required to change the momentum of an object, whether it is to increase it, decrease it, or to change its direction.

The concept of momentum is particularly important because, under certain situations, momentum is a conserved quantity. During the mid-seventeenth century, it had been observed that the momentum of colliding objects remain constant. No matter what the velocities and masses involved are, it is found that the total momentum before collision is the same as afterwards, whether the collision is head-on or not, as long as no net external force acts (Giancoli 1995).

A collision is a brief dynamic event consisting of the close approach of two or more particles, such as atoms, resulting in abrupt change of momentum, or exchange of energy (<http://www.answers.com/collision?cat=biz-fin>).

Impulse is equal to the total change in momentum. The following is the equation:

$$F\Delta t = p$$



The quantity on the left, the product of the force  $F$  times the change in time  $\Delta t$  over which the force acts, is the impulse. The concept of impulse is of help mainly with forces that act over a short time, as when a racket hits a ping pong ball.

There are two types of collisions: (1) elastic, and (2) inelastic. In an elastic collision, the total kinetic energy is conserved. Kinetic energy is represented as  $KE$ , and is computed by multiplying the mass of the object to the square of its velocity, then divided by two:

$$KE = \frac{1}{2} mv^2$$

In an elastic collision, on the other hand,  $KE$  is not conserved. The initial  $KE$ s of the two objects are equivalent to the final  $KE$ s plus the other form/s of energy produced (Giancoli 1995).

Collisions occur between the ball and table, and between the ball and the racket.

### B.3.2 Friction

Friction is a force that resists the relative motion or tendency to such motion of two bodies in contact (<http://www.answers.com/friction?cat=health>). The force of friction is solved by multiplying the coefficient of friction and the normal force exerted by the surface on the object:

$$F_{fric} = \mu F_N$$

The coefficient of friction ( $\mu$ ) will depend on the situation. If the object is stationary,  $\mu$  is equivalent to the *coefficient of static friction*. If the object's horizontal force exceeds the force of friction,  $\mu$  is equivalent to the *coefficient of kinetic or sliding friction*. The coefficient of static friction is almost always greater than the coefficient of kinetic friction, and it can never be less (Giancoli 1995).

Since table tennis involves a ball and a surface (table or paddle), only a small surface of the ball is contact with the table. And since the collision between the ball and

the surface takes a very small time frame, friction can only occur in that time frame, and there would not be enough time for the ball to slide on the surface. Thus, only static friction will be considered.

## C. Other Concepts

### C.1 Kinematics

Mechanics is the study of motion of objects and the related concepts of force and energy. It is divided into two parts: (1) kinematics, which is the description of how objects move, and (2) dynamics, which explains why the objects move as they do (Giancoli 1995).

Kinematics is well observed in the physical world. An object moving without rotating is called translational motion.

There are three major kinematics equations:

$$v_f = v_i + at$$

$$\Delta x = v_i t + \frac{1}{2} at^2$$

$$\Delta x = (v_f^2 - v_i^2) / 2a$$

The first equation is used in solving the final velocity of an object moving in a straight line.  $v_f$  is the final velocity,  $v_i$  is the initial velocity,  $a$  is the acceleration, and  $t$  is the time taken. The second and third equations are used in finding the change in the distance ( $\Delta x$ ).

### C.2 Gravity

Falling bodies are acted upon by a force called gravity. This force causes the object to accelerate downwards.

During Galileo's time, it was widely believed that an object's speed as it falls depends on its mass. Galileo believed otherwise. He postulated that all bodies would fall with the same constant acceleration in the absence of air or other kind of resistance. He showed that the postulate predicts for an object falling from rest, the distance traveled will be proportional to the square of the time. The uniform acceleration is also known as the acceleration due to gravity, and is given a symbol of  $g$ . On earth, it is approximately equal to  $9.8 \text{ m/s}^2$ , downwards. For simplicity of some computations,  $g$  is assumed to be negative.

The weight of an object is equivalent to its mass multiplied to the acceleration due to gravity. In equation form:

$$W = mg$$

Like all falling bodies, the table tennis ball also falls towards the earth.

### C.3 Newton's Laws

Newton has derived three laws of motion. They are the following:

- (1) *Every body continues in its state of rest or of uniform speed in straight line unless it is compelled to change that state by net force acting on it.*
- (2) *The acceleration of an object is directly proportional to the net force acting on it and is inversely proportional to its mass. The direction of the acceleration is in the direction of the applied net force.*
- (3) *Whenever one object exerts a force on a second object, the second object exerts an equal and opposite force on the first.*

The first law of motion is also known as the *Law of Inertia*. The measure of the inertia of a body is its mass. The second law is also known in the form of an equation:

$$\Sigma F = ma$$



The second law is one of the most fundamental relationships in physics. It describes quantitatively how forces affect motion. The third law is sometimes paraphrased as "*to every action there is an equal and opposite reaction.*" The third law is only applicable if the action force and the reaction force are acting as different objects (Giancoli 1995).

## **CHAPTER III**

### **METHODOLOGY**

#### **A. Overview of the Methodology**

The study aims to develop a computer software program that can simulate the path of a spinning lightweight spherical body (table tennis ball) and give realistic physics-based results rendered in 3D graphics. The program will be written and compiled in Visual C++ compiler with support from the Windows and DirectX APIs.

The program will have three major parts:

- (1) The Physics Engine – a collection of physics and math functions. It will be divided into two files: the CPP file which will contain the function definitions and the Header file which will contain the function prototypes/declarations and necessary data structures.
- (2) The Graphics Engine – a collection of graphics functions and sound and input wrapper functions. It will be adapted from Andre Lamothe's graphics engine that came in a CD along with his book *Tricks of the 3D Game Programming Gurus*.
- (3) The Shell Program – the main program that have access to both graphics and physics engine, thus integrating the two engines. The shell program will be written in a single CPP file. It will define how the program will look like and how it will run.

Each part will be tested after development of each one to minimize error at the end of the development of the final program.

## **B. Methods**

### **B.1 Gathering of Materials and Physics Concepts**

a. The internet was searched for different physics concepts governing table tennis.

The results were the following:

#### **Aerodynamics**

- Streamlines and Bernoulli's Principle
- Viscosity
- Boundary Layer
- Drag Force

#### **Momentum, Collisions and Impulse**

#### **Friction**

b. A personal computer (PC) installed with Visual C++ compiler and DirectX software development kit (SDK) will be used.

c. A royalty-free graphics engine that will be the basis of the study's own graphics engine will be taken from the CD that came with Andre Lamothe's Tricks of the 3D Game Programming Gurus book.

## B.2 Development of the Physics Engine

The physics engine will be built so that it will handle all the physics computations of the program. Some of these computations may be as simple as vector addition or as complex as computing for the Magnus Force.

The physics engine will be taking inputs defined by the shell program. Inputs will be initially given by the user, in turn processed by the shell program and passed to the physics engine. The inputs are the initial velocity and orientation, initial spin velocity and orientation, and position of the ball. The initial velocity and spin velocity are magnitudes, and the orientations are angles, where for the velocity, there are two angles.

The functions in the engine will involve the following computations:

### B.2.1 Drag Force

$$F_d = \frac{1}{2} \rho A v^2 C_d$$

where,  $\rho$  is the density of the fluid,  $A$  is the area of the object,  $v$  is the velocity, and  $C_d$  is the drag coefficient.

### B.2.2 Vectors

Addition

$$\mathbf{u} + \mathbf{v} = \langle u_x + v_x, u_y + v_y, u_z + v_z \rangle$$

Subtraction

$$\mathbf{u} - \mathbf{v} = \langle u_x - v_x, u_y - v_y, u_z - v_z \rangle$$

Length

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

## Cross Product

$$\mathbf{u} \times \mathbf{v} = \langle u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - v_x u_y \rangle$$

The formulas will be placed in separate functions with parameters. Then the function will use local variables to avoid unintended manipulation of other data. To apply the equations above, the following format will be observed:

```
type func_nam(func_params)
{
    operations
}
```

Where type is the variable type of the return value, func\_name is the name of the function, func\_params is/are the parameters, and the operations is/are the operations that will be performed on the values of the parameters.

As an example, here is a pseudocode for the addition of 2d vectors:

```
VEC2D add_vec2d(VEC2D v1, VEC2D v2)
{
    VEC2D temp;

    temp.x = v1.x + v2.x;

    temp.y = v1.y + v2.y;

    return(temp);
}
```



### B.2.3 Structures

Structures are variable type definitions that support public access member variables. An example of a structure is the following:

```
typedef VEC2D  
  
{  
  
    float x,y;  
  
}
```

The example structure defines the type VEC2D. Any variable with type VEC2D has the following sub-variables x and y unique only to the variable.

### B.3 Development of the Graphics Engine

The graphics engine will be adapted from that of Andre Lamothe that came with his book Tricks of the 3D Game Programming Gurus. It can render 3d objects on a 2d screen by performing transformations. The engine can also handle sound files and keyboard and mouse inputs. There will be modifications such as integration of multiple files into one and removing unneeded code.

### B.4 Development of the Shell Program

The shell program will have five main functions:

- (1) **WinMain()** – All windows Program must have this function. It handles the initializations of the window, registering the class, and the main program loop, which is responsible for throwing messages from Windows to the WinProc()

function, and making the call to the `Prog_Main()` function, updating the screen at a rate of 30 – 60 frames per second (fps).

- (2) **WinProc()** – This function will receive messages thrown by the `WinMain()` function which are not needed by the simulation. It will then decide what to do with the messages.
- (3) **Prog\_Init()** – This will handle all initializations not related to Windows but needed by the program. These initializations involves file loading, setting up of lookup tables, and other necessary initializations.
- (4) **Prog\_Shutdown()** – This function cleans up the resources taken and used by the program like releasing the DirectX interfaces.
- (5) **Prog\_Main()** – This function makes the calls to the two engines. This also decides what to do with gathered inputs. The flow of the simulation process will also be found here. Also, the interface of the program is defined here.

### **B.5 Testing of the Final Program**

The entire program will be tested. If the results are physically realistic, the next step will be taken. In case of an error, a bug, or a glitch, the part of the program that committed the error will be rewritten.

## CHAPTER IV

### RESULTS AND DISCUSSION

#### A. Results

The program did not work as expected. It worked without error on the computer unit used for the duration of the research, but when tested on a different unit, it closed and produced an error message. The program was already simplified to compute for the drag force only instead of the entire effects of spin. It turned into a software that shows the basics of a bouncing ball without spin.

The graphics of the program produced a wire frame 3-dimensional simulation of the ball instead of the intended output of a solid 3d ball model.

The physics computation was off. The effect of gravity on the ball was unrealistic and can be considered as completely wrong. The drag computation was plausible.

When the program was run, it takes the default value for the velocity of the ball and computes the drag force each iteration of the main loop of the program. The following handles the main loop of the program:

```
while(1)
{
    if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))
    {
        // test if this is a quit
        if (msg.message == WM_QUIT)
            break;

        // translate any accelerator keys
        TranslateMessage(&msg);
```



```
// send the message to the window proc  
DispatchMessage(&msg);  
} // end if  
  
// main game processing goes here  
Game_Main();  
} // end while
```

The computation were made in the `Game_Main()` function.

## B. Discussions

The effect of spin was not implemented in the study. It was supposed to be the main feature of the program. Friction computations were not made in the program. Drag force was the only physics computation that was retained. The separation of the computation of the physics aspect into another file made the code more complex. The computations were instead merged into the shell program code. It was placed inside the main loop. The graphics engine was maintained in separate files. It only has support for wire frame graphics in contrast to the expected solid graphics.

```
Load_OBJECT4DV1_3DSASC(&ball, "ball.obj", &vsacle, &vpos, &vrot);
```

This code loads the file `ball.obj` which contains the ball model data into the ball object pointer. `&vsacle`, `&vpos`, `&vrot` are pointers to the scale, position and rotation vectors, respectively.

The code above is defined below:

```
int Load_OBJECT4DV1_3DSASC(OBJECT4DV1_PTR obj, char *filename,
                          VECTOR4D_PTR scale, VECTOR4D_PTR pos,
                          VECTOR4D_PTR rot, int vertex_flags)
{
    CPARSERV1 parser;
    char seps[16];
    char token_buffer[256];
    char *token;
    int r,g,b;
    memset(obj, 0, sizeof(OBJECT4DV1));
    obj->state = OBJECT4DV1_STATE_ACTIVE | OBJECT4DV1_STATE_VISIBLE;
    if (pos)
    {
```

```
    obj->world_pos.x = pos->x;
    obj->world_pos.y = pos->y;
    obj->world_pos.z = pos->z;
    obj->world_pos.w = pos->w;
}
else
{
    obj->world_pos.x = 0;
    obj->world_pos.y = 0;
    obj->world_pos.z = 0;
    obj->world_pos.w = 1;
}
if (!parser.Open(filename))
{
    Write_Error("Couldn't open .ASC file %s.", filename);
    return(0);
}
while(1)
{
    if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
    {
        Write_Error("Image 'name' not found in .ASC file %s.",
filename);
        return(0);
    }
    if (parser.Pattern_Match(parser.buffer, "["Named"] ['object:']"))
    {
        strcpy(token_buffer, parser.buffer);
    }
}
```



```

strcpy(seps, "\\");
strtok( token_buffer, seps );
token = strtok(NULL, seps);
strcpy(obj->name, token);
Write_Error("\nASC Reader Object Name: %s", obj->name);
break;
}
}
while(1)
{
if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
{
Write_Error("'Tri-mesh' line not found in .ASC file %s.",
filename);
return(0);
}
if (parser.Pattern_Match(parser.buffer, ["Tri-mesh,']
['Vertices:'] [i] ['Faces:'] [i]"))
{
obj->num_vertices = parser.pints[0];
obj->num_polys = parser.pints[1];
Write_Error("\nASC Reader Num Vertices: %d, Num Polys: %d",
obj->num_vertices, obj->num_polys);
break;
}
}
while(1)
{

```

```

if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
{
    Write_Error("\n'Vertex list:' line not found in .ASC file %s.",
filename);
    return(0);
}

if (parser.Pattern_Match(parser.buffer, "["Vertex"] ["list:"]"))
{
    Write_Error("\nASC Reader found vertex list in .ASC file %s.",
filename);
    break;
}
}

for (int vertex = 0; vertex < obj->num_vertices; vertex++)
{
    while(1)
    {
        if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
        {
            Write_Error("\nVertex list ended abruptly! in .ASC file %s.",
filename);
            return(0);
        }
        StripChars(parser.buffer, parser.buffer, ":XYZ");
        if (parser.Pattern_Match(parser.buffer, "["Vertex"] [i] [f] [f]
[f]"))
        {

```

```

obj->vlist_local[vertex].x = parser.pfloats[0];
obj->vlist_local[vertex].y = parser.pfloats[1];
obj->vlist_local[vertex].z = parser.pfloats[2];
obj->vlist_local[vertex].w = 1;

#define VERTEX_FLAGS_INVERT_X 1
#define VERTEX_FLAGS_INVERT_Y 2
#define VERTEX_FLAGS_INVERT_Z 4
#define VERTEX_FLAGS_SWAP_YZ 8
#define VERTEX_FLAGS_SWAP_XZ 16
#define VERTEX_FLAGS_SWAP_XY 32
#define VERTEX_FLAGS_INVERT_WINDING_ORDER 64

float temp_f;

if (vertex_flags & VERTEX_FLAGS_INVERT_X)
    obj->vlist_local[vertex].x=-obj->vlist_local[vertex].x;

if (vertex_flags & VERTEX_FLAGS_INVERT_Y)
    obj->vlist_local[vertex].y=-obj->vlist_local[vertex].y;

if (vertex_flags & VERTEX_FLAGS_INVERT_Z)
    obj->vlist_local[vertex].z=-obj->vlist_local[vertex].z;

if (vertex_flags & VERTEX_FLAGS_SWAP_YZ)
    SWAP(obj->vlist_local[vertex].y, obj->vlist_local[vertex].z,
temp_f);

if (vertex_flags & VERTEX_FLAGS_SWAP_XZ)
    SWAP(obj->vlist_local[vertex].x, obj->vlist_local[vertex].z,
temp_f);

if (vertex_flags & VERTEX_FLAGS_SWAP_XY)
    SWAP(obj->vlist_local[vertex].x, obj->vlist_local[vertex].y,
temp_f);

```



```

Write_Error("\nVertex %d = %f, %f, %f, %f", vertex,
            obj->vlist_local[vertex].x,
            obj->vlist_local[vertex].y,
            obj->vlist_local[vertex].z,
            obj->vlist_local[vertex].w);

if (scale)
{
    obj->vlist_local[vertex].x*=scale->x;
    obj->vlist_local[vertex].y*=scale->y;
    obj->vlist_local[vertex].z*=scale->z;
}

break;
}
}

Compute_OBJECT4DV1_Radius(obj);
Write_Error("\nObject average radius = %f, max radius = %f",
            obj->avg_radius, obj->max_radius);

while(1)
{
    if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
    {
        Write_Error("\n'Face list:' line not found in .ASC file %s.",
filename);

        return(0);
    }

    if (parser.Pattern_Match(parser.buffer, "'Face' ['list:']"))
    {

```

```

Write_Error("\nASC Reader found face list in .ASC file %s.",
filename);
    break;
}
}

int poly_surface_desc = 0; case
int poly_num_verts    = 0;
char tmp_string[8];
for (int poly=0; poly < obj->num_polys; poly++)
{
    while(1)
    {
        if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
        {
            Write_Error("\nface list ended abruptly! in .ASC file %s.",
filename);
            return(0);
        }
        StripChars(parser.buffer, parser.buffer, ":ABC");
        if (parser.Pattern_Match(parser.buffer, ["'Face' [i] [i] [i]
[i]"]))
        {
            if (vertex_flags & VERTEX_FLAGS_INVERT_WINDING_ORDER)
            {
                poly_num_verts    = 3;
                obj->plist[poly].vert[0] = parser.pints[3];
                obj->plist[poly].vert[1] = parser.pints[2];
                obj->plist[poly].vert[2] = parser.pints[1];
            }
        }
    }
}

```

```

    )
else
    {
        poly_num_verts          = 3;
        obj->plist[poly].vert[0] = parser.pints[1];
        obj->plist[poly].vert[1] = parser.pints[2];
        obj->plist[poly].vert[2] = parser.pints[3];
    }

    obj->plist[poly].vlist = obj->vlist_local;
    break;
}

}

while(1)
{
    if (!parser.Getline(PARSER_STRIP_EMPTY_LINES |
PARSER_STRIP_WS_ENDS))
    {
        Write_Error("\nmaterial list ended abruptly! in .ASC file %s.",
filename);
        return(0);
    }

    ReplaceChars(parser.buffer, parser.buffer, ":\\"rgba", ' ');

    if (parser.Pattern_Match(parser.buffer, "[i] [i] [i]"))
    {
        r = parser.pints[0];
        g = parser.pints[1];
        b = parser.pints[2];

        if (screen_bpp==16)
            {

```



```

    SET_BIT(obj->plist[poly].attr, POLY4DV1_ATTR_RGB16);
    obj->plist[poly].color = RGB16Bit(r, g, b);
    Write_Error("\nPolygon 16-bit");
}
else
{
    SET_BIT(obj->plist[poly].attr, POLY4DV1_ATTR_8BITCOLOR);
    obj->plist[poly].color = RGBto8BitIndex(r, g, b, palette, 0);
    Write_Error("\nPolygon 8-bit, index=%d", obj-
>plist[poly].color);
}

    POLY4DV1_ATTR_SHADE_MODE_GOURAUD);
    POLY4DV1_ATTR_SHADE_MODE_PHONG);
    SET_BIT(obj->plist[poly].attr, POLY4DV1_ATTR_SHADE_MODE_FLAT);
    obj->plist[poly].state = POLY4DV1_STATE_ACTIVE;
    break;
}
}

Write_Error("\nPolygon %d:", poly);

Write_Error("\nSurface Desc = [RGB]=[&d, &d, &d], vert_indices
[&d, &d, &d]", r, g, b, obj->plist[poly].vert[0], obj-
>plist[poly].vert[1], obj->plist[poly].vert[2]);
}

return(1);
}

```

The following code renders the ball model on the screen.

```

DDraw_Lock_Back_Surface();
Draw_OBJECT4DV1_Wire16(&ball, back_buffer, back_lpitch);

```

```
DDraw_Unlock_Back_Surface();
```

33

```
Draw_OBJECT4DV1_Wire16(&ball, back_buffer, back_lpitch) is defined:  
void Draw_OBJECT4DV1_Wire16(OBJECT4DV1_PTR obj, UCHAR *video_buffer,  
int lpitch)
```

```
{  
// this function renders an object to the screen in wireframe,  
// 16 bit mode, it has no regard at all about hidden surface removal,  
// etc. the function only exists as an easy way to render an object  
// without converting it into polygons, the function assumes all  
// coordinates are screen coordinates, but will perform 2D clipping  
// iterate thru the poly list of the object and simply draw  
// each polygon  
for (int poly=0; poly < obj->num_polys; poly++)  
{  
// render this polygon if and only if it's not clipped, not culled,  
// active, and visible, note however the concept of "backface" is  
// irrelevant in a wire frame engine though  
if (!(obj->plist[poly].state & POLY4DV1_STATE_ACTIVE) ||  
    (obj->plist[poly].state & POLY4DV1_STATE_CLIPPED) ||  
    (obj->plist[poly].state & POLY4DV1_STATE_BACKFACE) )  
    continue; // move onto next poly  
// extract vertex indices into master list, rember the polygons are  
// NOT self contained, but based on the vertex list stored in the  
object  
// itself  
int vindex_0 = obj->plist[poly].vert[0];  
int vindex_1 = obj->plist[poly].vert[1];  
int vindex_2 = obj->plist[poly].vert[2];
```

```
// draw the lines now
Draw_Clip_Line16(obj->vlist_trans[ vindex_0 ].x, obj->vlist_trans[
vindex_0 ].y,
obj->vlist_trans[ vindex_1 ].x, obj->vlist_trans[
vindex_1 ].y,
obj->plist[poly].color,
video_buffer, lpitch);

Draw_Clip_Line16(obj->vlist_trans[ vindex_1 ].x, obj->vlist_trans[
vindex_1 ].y,
obj->vlist_trans[ vindex_2 ].x, obj->vlist_trans[
vindex_2 ].y,
obj->plist[poly].color,
video_buffer, lpitch);

Draw_Clip_Line16(obj->vlist_trans[ vindex_2 ].x, obj->vlist_trans[
vindex_2 ].y,
obj->vlist_trans[ vindex_0 ].x, obj->vlist_trans[
vindex_0 ].y,
obj->plist[poly].color,
video_buffer, lpitch);

// track rendering stats
#ifdef DEBUG_ON
debug_polys_rendered_per_frame++;
#endif
} // end for poly
} // end Draw_OBJECT4DV1_Wire16
```

The next code computes for the drag force:

```
DRAGZ = DRAGC * AIRD * vel_ball * vel_ball * ball_area / 2;
vel_ball = vel_ball - (DRAGZ/ball_mass) * 0.16;
ball_posz = ball_posz + (vel_ball * 0.16);
```

DRAGC is the drag coefficient for a smooth spherical body which has the value of 0.5. AIRD is the density of air with a value of 1.229. vel\_ball is the current instantaneous velocity of the ball, ball\_area is the cross-sectional area of the ball, ball\_mass is its mass, and ball\_posz is its position along the z-axis.

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg, WPARAM wparam,
                             LPARAM lparam)
```

```
{
    PAINTSTRUCT ps;
    HDC hdc;

    switch(msg)
    {
        case WM_CREATE:
            {
                return(0);
            } break;

        case WM_PAINT:
            {
                hdc = BeginPaint(hwnd, &ps);

                EndPaint(hwnd, &ps);

                return(0);
            } break;

        case WM_DESTROY:
            {
```



```
        postQuitMessage(0);  
        return(0);  
    } break;  
    default:break;  
}  
return (DefWindowProc(hwnd, msg, wparam, lparam));  
}
```

This code handles all the processes that are not needed by the program but may be needed by Windows. This manages the data sent to it and decides whether to throw the data to the default Windows Process function or use the data to quit or refresh the program.

Most of the intended computations and expected outputs of the program were not acquired.

## CHAPTER V

### SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

#### A. Summary

The aim of this study is to create a computer program that can simulate the path of a lightweight spherical body, specifically, a table tennis ball, with plausible physics-based results. The program was made in three parts, the graphics engine which handles the graphics, the physics engine which manages the physics computation, and the main shell program which integrates the two engines and handles the window related functions.

The program displays a full screen 3D graphical wire frame animation of the path of the ball with the given initial velocity and the default values. It also displays on-screen the current velocity and forces acting on the ball.

#### B. Conclusions

Creating a program that can simulate the path of a lightweight spherical body with plausible physics-based result rendered in 3D graphics is attainable given that the maker of the program has enough knowledge on the subject matter and the programming language being used and has enough time dedicated in making program.

#### C. Recommendations

Future studies similar to this study are recommended to apply spin on the physics computations and upgrading the wire frame graphics to solid graphics, and also using the Windows GUI interface instead of full screen. If possible, making the program cross platform is also recommended.

## Websites

- Anonymous. Bouncing Ball. Available: <http://web.kellagous.com/ecrits/000858>
- Anonymous. C++. Available: <http://en.wikipedia.org/wiki/C++>.
- Anonymous. DirectX. Available: <http://en.wikipedia.org/wiki/DirectX>.
- Anonymous. Drag (physics). Available:  
[http://en.wikipedia.org/wiki/Drag\\_\(physics\)](http://en.wikipedia.org/wiki/Drag_(physics)).
- Anonymous. Magnus Effect. Available:  
[http://en.wikipedia.org/wiki/Magnus\\_effect](http://en.wikipedia.org/wiki/Magnus_effect).
- Anonymous. Rigid Body Dynamics. Available:  
<http://www.cs.brown.edu/courses/gso07/lect/sim/rigid/slide.html>.
- Anonymous. Simulation. Available: <http://en.wikipedia.org/wiki/Simulation>.
- Carini, J.. Effects of Spin on the Motion of a Ball. 26 July 2005. Available:  
<http://carini.physics.indiana.edu/E105/spinning-balls.html>.
- Filippone, A., 2004. Aerodynamic Database: Drag Coefficients. Available:  
<http://aerodyn.org/Drag/tables.html>.
- NASA Glenn Research Center. Ideal Lift of a Spinning Ball. Available:  
<http://www.grc.nasa.gov/WWW/K-12/airplane/beach.html>.
- Hsieh, C., Chang, C.. Rigid Body Simulation. 12 December 2006. Available:  
<http://www-scf.usc.edu/~chiaminc/>.
- Letts, G.. The Basic Physics and Mathematics of Table Tennis / Ping Pong.  
Available:  
[http://tabletennis.about.com/od/beginnersguide/a/physics\\_mathsTT.htm](http://tabletennis.about.com/od/beginnersguide/a/physics_mathsTT.htm).
- Letts, G.. The Physics of Spin in Table Tennis. Available:  
[http://tabletennis.about.com/od/basicconcepts/ss/spin\\_in\\_tt.htm](http://tabletennis.about.com/od/basicconcepts/ss/spin_in_tt.htm).
- Lewis, D.. CS448A Final Project: Rigid Body Simulation. Available: <http://www-cs-students.stanford.edu/~dalewis/cs448a/rigidbody.html>.

**Electronic Journal Article**

Baraff, D.. SIGGRAPH90 Course Notes Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation. August 1990.

Baraff, D.. SIGGRAPH91 Course Notes Coping with Friction for Non-penetrating Rigid Body Simulation. July 1991.

Berger, M.. Physics of Baseball: A Study of Pitching. 1 May 2000.

Guendelman, E., Bridson, R., Fedwik, R.. Nonconvex Rigid Bodies with Stacking.