Squash Ball Simulation In OpenGL

RUI BI

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

September 2005

Your file Votre référence
ISBN: 0-494-10281-0
Our file Notre référence
ISBN: 0-494-10281-0

Abstract

Squash Ball Simulation In OpenGL

Rui Bi

In this thesis, we show how to construct a 3D model of a squash ball, analyze dynamics and impact force, and simulate the deformation of the ball in the real world. The goal of this project is to achieve a realistic behavior of a squash ball using physical calculations. Some algorithms, simulation results, user-interface design and future work are presented in this report.

Acknowledgements

This work would not have been possible without the guidance and encouragement of my supervisor, Prof. Peter Grogono. Thank you Peter for helping to set up the project in the first place, providing resources and offering direction throughout the course of this work.

Also, I would like to express my gratitude to all the people who have provided enthusiastic support for this project.

Table of Contents

LIST OF FIGURES

1. Introduction

In this thesis, we describe a 3D simulation of a squash game coded using OpenGL and C++. The thesis focuses on the changing shape of the squash ball rather than the complete game. The goal of the research of this project is to simulate the behavior of a squash ball and to visualize the simulation by means of physical calculations using object-oriented technique with OpenGL.

1.1 Motivation

In the 3D graphics world, deformation physics is applied widely. It is an advanced technique needed to create a realistic 3D graphic environment. This project is concerned with modeling a physics-based squash ball and providing controls that enable the user to interact with the simulation.

1.2 Thesis Outline

The organization of this thesis is as follows. The second chapter generally introduces related work and technical background including OpenGL fundamentals. The third chapter focuses on the design methods, physics-based modeling, as well as physical theories and computations. The fourth chapter describes the implementation in details, including class definitions, GLUI (GLUT-Based User Interface library) controls and solutions to some problems. The fifth chapter reports some experimental

results and analyzes the results. The last chapter concludes with successes and limitations of this project and suggestions for future work.

2. Background

In this chapter, we describe squash games, related work and basic technical background of this project.

2.1 Related Work

There are two kinds of objects in ball game simulations: rigid and deformable. Deformable objects are harder to simulate than rigid ones. For some games such as billiards and pool, the balls are so hard that their deformation can be ignored; while for others such as squash, the deformation of the balls are good to be simulated. Squash is one of a large family of games in which an elastic ball is struck with an implement. This family includes basketball, racquetball, squash, and so on.

2.1.1 Bouncing Ball Simulations

In this section, we introduce some research and work in deformable ball simulations.

2.1.1.1 Air-filled Ball

When a ball bounces on the court, its speed, acceleration, and form change. M. James Bridge used a 2D Java applet to simulate the bounce of an air-filled ball, like a basketball [13][14]. The program demonstrates how the drop height, which is

proportional to the impact velocity and pressure, influences the response of the ball after it impacts the ground. This project is limited in two ways. First, the part of the ball not in contact with the ground remains spherical, whereas a real ball distorts under its own weight and momentum. Second, the ball does not oscillate after impact because the applet does not model the ball's elasticity. In the real world, this part of ball also squeezes in and rebounds out. A racquetball's behavior is similar to a basketball's, but racquetballs are softer and therefore more deformable.

2.1.1.2 Pressure Soft Ball

Maciej Matyka implemented a 2D pressure softball model using OpenGL [15]. His paper "How to implement a pressure soft body model" contains detailed description of how to start the simplest program for simulation of soft bodies [15]. He explained spring force and damping force calculation using Hooke's law, which we will see again in section 3.2.

In another paper, "Pressure model of softball simulation" [16], Matyka introduced a common technique for calculation of soft objects volume - bounding volume: simply define the maximum and minimum positions in X and Y directions of all body points, the bounding box (rectangle in two dimensions) gives first approximation of the body volume.

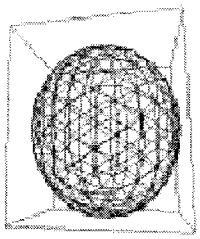


FIGURE 2-1 BOUNDING BOX OF A BALL

As shown in figure 2-1, the bounding box can be an approximation of the ball. Bounding volumes are not very accurate. They can be used for general volume changes, while we need more precise volume for pressure calculation, which we will describe in section 3.3.

2.1.1.3 3D Deformable Ball Simulation

Recently, Jessica Fitch and J. Reed Walker posted their work "Deformable ball simulation", developed using OpenGL [17]. They used point cloud presentation, a 3D modeling technique to generate a 3D ball. They used the principle of linear impulse and momentum to get the contact force; then, calculated the displacement of the ball. They assumed the volume is preserved during the deformation, so they applied an algorithm to distribute the volume difference to per visible slice of the ball. In reality, volume does change during deformation.

2.1.2 Deformable Models

In this section, we introduced some previous work on deformable modeling suitable for animating flexible objects.

Demetri Terzopoulos, John Platt, Alan Barr and Kurt Fleischer described how to model elasticity deformable materials such as rubber in their paper "Elastically deformable models" [18]. They employed elasticity theory to construct differential equations that model the behavior of non-rigid curves, surfaces, and solids as a function of time. They described active elastically deformable models: they respond in a natural way to applied forces, constraints, ambient media, and impenetrable obstacles.

In another article, "Dynamic deformation of solid primitives with constraints" [19], Demetri Terzopoulos developed a systematic approach to deriving dynamic models from parametrically defined solid primitives, global geometric deformations, and local finite element deformations. Even though their kinematics is stylized by the particular solid primitive used, the models behave in a physically correct way with prescribed mass distributions and elasticities.

Kazuya G. Kobayashi and Katsutoshi Ootsubo described a method of free-form deformation in their paper "t -FFD: free-form deformation by using triangular mesh" [20]. An original shape of large-scale polygonal mesh or point-cloud is deformed by

using a control mesh, which is constituted of a set of triangles with arbitrary topology and geometry, including the cases of disconnection or self-intersection. This method works on a simple mapping mechanism. First, each point of the original shape is parameterized by the local coordinate system on each triangle of the control mesh. After modifying the control mesh, the point is mapped according to each modified triangle. Finally, the mapped locations are blended as a new position of the original point, then a smoothly deformed shape is achieved.

2.1.3 Racquetball Bounce

We cannot see the distortion of a racquetball accurately, while a special high-speed camera can. Figure 2-2 shows the gradual deformation of a racquetball hit the ground and rebound. These video clips were taken via high-speed camera. It takes time to rebound back to the original round shape and constant speed because of the resilient and damping factors of the material. The two scenarios in this project - the racquet hitting the ball then the ball rebounding from the front wall, and the ball dropping then rebounding from the floor, imitate these motions in the real world.

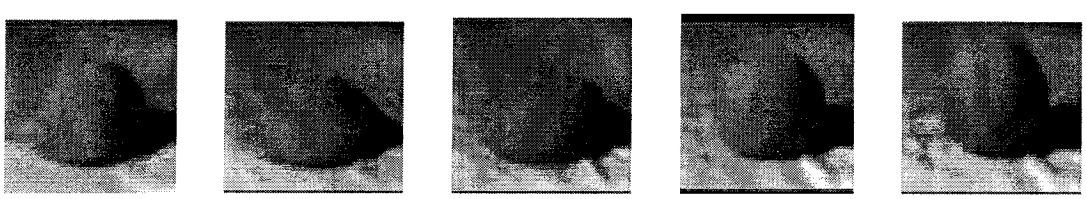


FIGURE 2-2 BALL BOUNCE FROM THE GROUND

2.2 Squash Overview

Compared with a racquetball, a squash ball is smaller and less bouncy. The diameter of the ball, measured perpendicular to the seam, must be between 39.5mm and 40.5mm [10]. The diameter of the squash ball in this project is 40 mm, a bit less than 2 inches. This project demonstrates a standard single court with the following dimensions [9]:

	Length	Width
Court	9.754m (≈32 feet)	6.402m (≈21 feet)

	Height
Front Wall	4.572m (≈ 15 feet)
Back Wall	2.134m (≈ 7 feet)

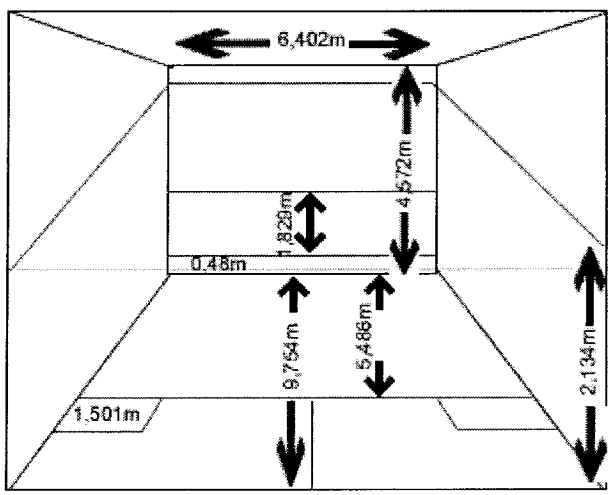


FIGURE 2-3 SQUASH COURT

Since the purpose of the simulation is to show the deformation of the ball, the program exaggerates the size of the ball and the racquet, and renders the walls of the court transparently. The second effect is actually realistic, because U.S. and English open squash matches are played in transparent courts so that the audience can see the action.

2.3 Technical Background

Before discussing the design of this project, we review OpenGL fundamentals, Object-Oriented Programming and the relation between graphics and physics.

2.3.1 3D Graphics Concepts

OpenGL consists of two libraries: the Graphics Library (GL), the Graphics Library Utilities (GLU). The OpenGL Utility Toolkit (GLUT) implements a simple window application programming interface (API) for OpenGL.

GLUT provides the functionality necessary to transfer the OpenGL frame buffer to a window on the screen. GLUT programs are platform-independent: a GLUT program can be compiled and run on a unix workstation, a PC running Windows, or a Mac, with substantially the same results [1].

2.3.1.1 Modeling Transformations

We present an entire squash court. A user may rotate, translate and scale the whole scene to get the best visual effects. These operations use modeling transformations. Modeling transformations change the coordinates of a point from (x_0, y_0, z_0) to (x, y, z). The common kinds of modeling transformations are [1]:

Translation: $x = x_0 + a$

$y = y_0 + b$

$z = z_0 + c$

Scaling: $x = r\,x_0$

$y = s\,y_0$

$z = t\,z_0$

Rotation: $x = x_0 \cos\theta - y_0 \sin\theta$

$y = x_0 \sin\theta + y_0 \cos\theta$

$z = z_0$

(This is a rotation about the Z-axis. Equations for rotations about the other axes are similar.)

By using homogeneous coordinates, we can use matrices to represent all three kinds of transformation. 4×4 matrices are required for these 3D transformations.

The OpenGL commands for these transformations are glTranslate(), glRotate(), and glScale(). All these commands generate equivalent translation, rotation and scaling matrices; then these matrices are multiplied with the current matrix.

2.3.1.2 Viewing Transformations

A viewing transformation locates and changes orientation of the viewport, and positions the camera at the origin. To achieve the complete scene or partial scene in a final picture, the camera and the model must be separated. The program can do this by applying a transformation that moves the camera away from the model. For every such transformation, there is a corresponding inverse transformation that moves the model. Thus, modeling transformations and camera transformations are complementary [2].

To construct a viewing transformation, we can also use gluLookAt (). This routine simulates "looking at" the scene by defining the camera or eyes position and direction towards the scene. gluLookAt () is useful because it constructs a transformation that is hard to compute. Without it, you have to translate away from the model, rotate to get a particular point at the center of the image, and then twist to get the 'up' vector vertical.

2.3.2 Object-Oriented Language

C++ is an object-oriented Language used widely in 3D objects design and implementation. In Object-Oriented programming (OOP), objects represent real-world objects with the properties we choose to manipulate. There are a lot of advantages using OOP in this project:

- The encapsulation mechanism keeps the data private.
- It makes easier to build up models, organize and expand the system.
- The code is concise, readable, and easier to write.

OpenGL is a C Application Programming Interface (API). Using OOP techniques, we build classes that call the basic GL functions. These classes are described in chapter 3.

2.3.3 Physical Environment

As computational power increases, users and applications are demanding increasing levels of realism in these domains. This trend is particularly apparent in computer graphics where more sophisticated geometric shapes and physical objects are being modeled in the context of complex physical environments. In particular, the ability to model and manipulate deformable objects is essential to many applications [4].

2.3.3.1 Physics-based Modeling

Physics-based technique is based on continuum mechanics. It involves the effects of material properties, external and internal forces, and environmental constraints on objects.

We simulate a squash ball deformation in the following physical environment:

- Material – synthetic materials mixed with rubber, modeled using mass-spring
- Internal force – spring force, inside pressure
- External force – racquet impact force, gravitational force, air resistance

Another modeling technique, non-physics technique, uses purely geometric technique: individual or group of control points or shape parameters are manually adjusted for shape editing and design. Non-physics technique relies on the skill of the designer rather than on physical principles. Using Non-physics technique, deformation needs to be explicitly specified and the system has no knowledge about the nature of the objects being manipulated.

Compared with non-physics methods, physics methods are free from the limitation from the user's expertise. Assisted with geometry technique, physical principles and processes make it possible to compute the shapes and motions of deformation objects. In chapter 3, we describe what physical principles and how these physical principles are applied in this project.

2.3.3.2 Approximation of Graphics

Graphics is all approximation and illusion. Even with fast, modern processors, we cannot hope to simulate accurately every molecule of real objects. So we have to find approximations that don't take too long to compute. In graphics, what matters is the appearance.

3. Design

In this chapter, we describe how to create a 3D ball, define some physics laws and formulas applied in this project as well as related dynamic calculations, and explain the main classes.

3.1 3D Model

In 3D games design, balls are composed by several patches as polygons; they are triangles, trapezoids, pentagons, etc. These polygons can be used uniquely or jointly.

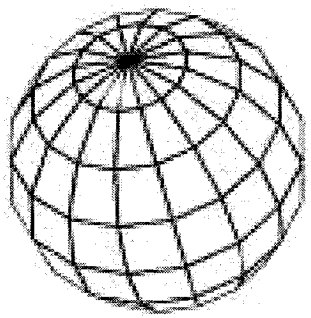


FIGURE 3-1 TRAPEZOID MESH

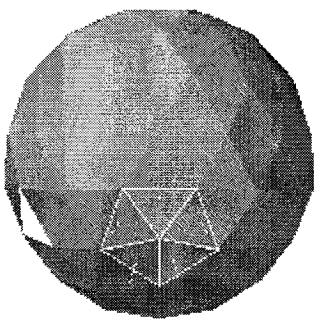


FIGURE 3-2 TRIANGLE MESH

The following table shows the difference between a triangle mesh and a trapezoid mesh in various aspects:

Triangle Mesh	Trapezoid Mesh
Each vertex connects 5 or 6 edges.	Each vertex connects 4 edges. Fewer edges simplifies spring force and normal calculations on each vertex.
It is not completely symmetrical: triangles have different size (smaller triangles are used where more details are needed). It is harder to construct.	It is simpler to construct a sphere using trapezoids: it is easy to find points on the latitudes and longitudes of a sphere; four adjacent points enclose a trapezoid.
It is close to symmetry and it is easy to render.	It is harder to render than triangles because trapezoids consist of four edges while triangles consist of three edges.
Triangles always lie in a plane.	Trapezoids may not lie in a plane.

3.1.1 Ball Frame

Similar to a globe divided in latitudes and longitudes, the surface of the simulated ball is divided into a number of horizontal rings with radius w and height z (called stacks in this project) and vertical equal rings that pass two poles (called slices in this projects).

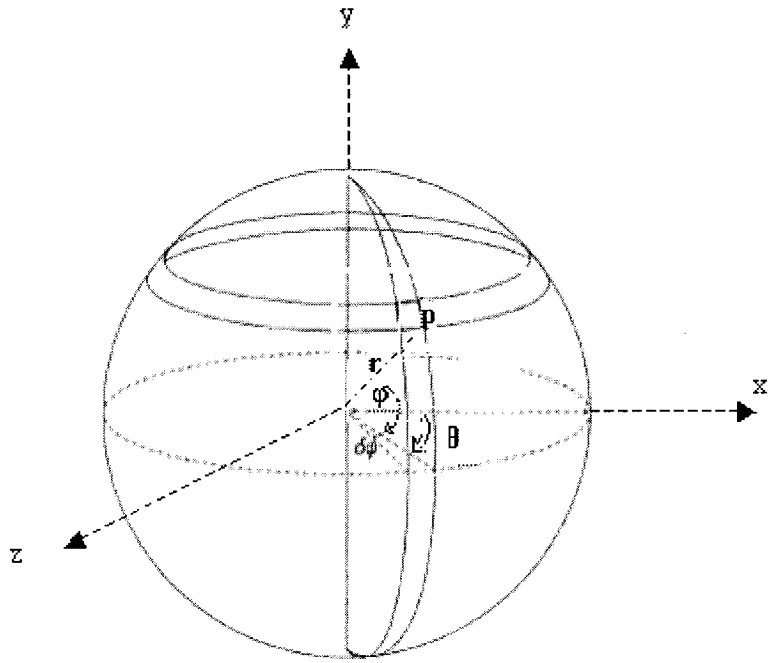


FIGURE 3-3 BALL'S CONSTRUCTION

As shown in figure 3-3, two adjacent slices and stacks enclose one trapezoid. There is a special case for two poles: the two rounds of patches around two poles are composed by triangles. Connected on two adjacent vertices on the same horizontal ring to the center of the ball, the angles $\delta\varphi$ are equal on all horizontal rings; the same is true for two adjacent vertices on the same vertical ring. Thus, given the radius, number of slices, and number of stacks, we can build a 3D ball using sine and cosine functions. Simultaneously, we store the positions of all the vertices in a two-dimensional array of stacks and slices for later normal vectors and dynamic computing.

The initial position of point P on the surface of the ball is (figure 3-3):

$$x = r \cos \varphi \cos \theta$$

$$y = r \sin \varphi$$

$$z = r \cos \varphi \sin \theta$$

With $0 \leq \varphi \leq \pi$, $0 \leq \theta < 2\pi$.

To obtain discrete points, we assume N is the number of stacks and M is the number of slices, use i = 0, 1, 2 . . . , N and j = 0, 1, 2. . . , M-1, then:

$$\varphi = i\pi / N$$

$$\theta = 2j\pi / M$$

3.1.2 Normal Vectors

A surface normal at a point on a surface is a vector orthogonal to the tangent plane at that point. By default, OpenGL assumes that the vector is a unit vector for lighting calculations, which is why it is best to use unit normals.

In this project, a squash ball is constructed of many small trapezoids whose edges are lines of latitude and longitude. We use average normals at the vertices, and OpenGL will interpolate colors across the trapezoids, giving the illusion of a curved surface. Newell's formula (figure 3-4) is a simple and robust way to compute the normal to a polygon since it takes all vertices into consideration. Newell's algorithm [3] is exact

if the points lie in a plane. It is also useful for curved surfaces due to the fact that it gives a good approximation even if the points are not exactly coplanar. In this project, Newell's formula is used to calculate the changing normals of points on the surface of the squash ball when it deforms.

$$N_x = \sum_{i=0}^{N-1} (y_i - y_{((i+1) \bmod N)})(z_i + z_{((i+1) \bmod N)})$$

$$N_y = \sum_{i=0}^{N-1} (z_i - z_{((i+1) \bmod N)})(x_i + x_{((i+1) \bmod N)})$$

$$N_z = \sum_{i=0}^{N-1} (x_i - x_{((i+1) \bmod N)})(y_i + y_{((i+1) \bmod N)})$$

FIGURE 3-4 NEWELL'S ALGORITHM

3.1.3 Spring Mesh

"Mass-spring model is a physics-based technique that has been used effectively for deformable objects" [4]. In order to model a squash ball deformation, the surface of a squash ball is modeled as point masses connected by springs in a lattice structure (figure 3-1).

Each point on surface joins four neighbor springs. The neighbors of P [i, j], in counterclockwise order are (figure 3-5), where i = 0, 1, 2 . . . , N and j = 0, 1, 2. . . , M-1, N is number of stacks and M is number of slices, then:

P1 = P [i - 1, j]

P2 = P [i, j - 1]

P3 = P [i +1, j]

P4 = P [i, j +1]

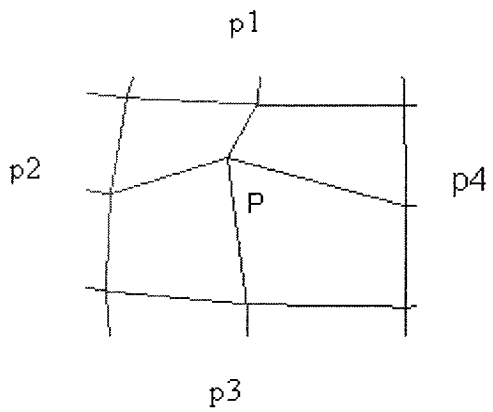


FIGURE 3-5 A PATCH OF SPRING MESH

If j = 0, use M-1 instead of j -1.

If j = M-1, use 0 instead of j +1.

There are some special cases:

- Most points have four neighbors. For the points near the two poles (i = 1 and i = N-1), one of their neighbors is the pole.
- Each pole has M neighbors.

At each point P [i, j], there is a normal vector n [i, j]. The normal can be calculated applying Newell's formula [3] to the point and its neighbors.

3.2 Dynamics

This section is focused on variety of quantities including force, acceleration, velocity and displacement, dynamic equations, and Newton's laws, which are used in describing the motion of a squash ball. All the quantities are represented by 3D vectors, which have both magnitude and direction. All the mathematical operations on these quantities are performed with or upon vectors.

3.2.1 Forces

A force is a push or pull acting upon an object as a result of its interaction with another object [6]. There are a variety of types of forces functioning on a squash ball: spring force, gravitational force, applied force, and air resistance.

3.2.1.1 Spring Force

The spring force is the force that results when a spring is compressed or stretched. Hooke's spring law determines the forces acting on two particles connected by a spring. Each spring has a rest length. If the spring length is greater than this length then the force acts to pull the two particles together; if the spring length is less than

the rest length then the force acts to repel the two particles. "For most springs, the magnitude of the force is directly proportional to the amount of stretch or compression" [6].

In this project, each spring has a natural length that is computed when the points lie on the undistorted ball. We write $natlen_k$ for the natural length and $actlen_k$ for the actual length of spring k. A spring produces equal and opposite forces on the points at its ends. Suppose spring S joins points P and Q, and u is the unit vector in the direction PQ (figure 3-6). Then:

Force on P = F_s u

Force on Q = -F_s u

Where $F_s = K_s (natlen_k - actlen_k)$. K_s is the spring coefficient (also called spring constant), the stiffness of the spring.

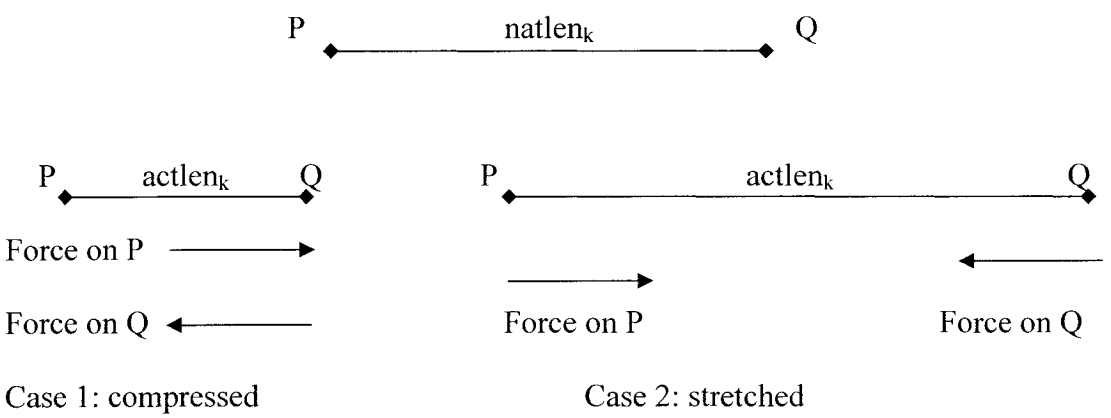


FIGURE 3-6 SPRING FORCE

Here is what should happen: initially, the spring forces should be zero because the springs are all at their natural length. If the excess pressure inside the ball is zero (pressure is explained in section 3.3) and the ball does not impact any objects, the ball should not deform. There are two aspects causing spring mesh to compress and stretch:

- Pressure - If the excess pressure inside the ball is positive, the ball should expand until the spring forces counteract the pressure. After expanding, it will contract again. In fact, it will continue oscillating forever if the system is undamped.

- Obstacles - When the ball hits the racquet, floor, and wall of the court, the impacting springs will be stretched or compressed from their equilibrium positions and influence all connecting springs.

3.2.1.2 Damping

Damping is the amount of energy absorbed by the spring as it bounces. Damping is necessary so that the spring doesn't keep oscillating forever. Damping can be modeled as force acting in the opposite direction to the velocity: $F_d = - K_d v$, where K_d is the damping constant.

The spring damping force depends on the difference in velocity of the two end points. Both of the spring and the damping forces act along the line defined by the two end

points. Given two end points a and b conntected by a spring, according Hooke's spring law, the force on point a is given by [5]:

$$f = -\left[k_s(\| x_a - x_b \| - r) + k_d (v_a - v_b). \frac{x_a - x_b}{\| x_a - x_b \|} \right] \frac{x_a - x_b}{\| x_a - x_b \|}$$

Where Xa and Xb are the positions of particles a and b, Va and Vb are the velocities of a and b, and r is the rest length between a and b. We have already discussed the part "Ks ($||X_a-X_b||$ - r)", the spring force, in section 3.2.1.1. The other part "K_d (V_a-V_b) • (X_a-X_b)/ ($|| X_a-X_b ||$)" is the damping force between two particles.

Since the squash ball is composed of a mass-spring mesh, for each point, the differences in the velocities of all four adjacent points should be considered for damping force calculation:

F_d = -K_d ((V [i, j] – V [i+1, j])
+ (V [i, j] – V [i-1, j])
+ (V [I, j] – V [i, j-1])
+ (V [i, j] – V [i, j+1]))

V [i, j] is the velocity of point P [i, j], and V [i+1, j], V [i-1, j], V [i, j-1], V [i, j+1] are velocities of its four neighbor points.

3.2.1.3 Gravity

The force of gravity accelerates the squash ball when it drops towards the floor and decelerates it when it rebounds upwards. Weight is the force generated by the gravitational attraction of the earth on any object. By definition, weight is the mass of an object times the gravitational acceleration:

$$F_g = mg$$

Where m is the mass of the object and $g \approx 9.8\ m/s^2$ is the acceleration due to gravity.

3.2.1.4 Applied Force

An applied force is a force that is applied to an object by a person or another object [6]. When a player serves, the translated force from the player due to impact acting upon the squash ball is an applied force.

Momentum can be defined as "mass in motion" [7]. It is the product of the object's mass m and velocity v, or mv. A force acting for a given amount of time will change an object's momentum [7]. The equation can be written as a consequence of Newton's second law (explained in 4.2.2.2):

$$F\Delta t \approx m\ \Delta v$$

The following shows how this equation is derived from the Newton's second law: Newton's Second Law says that force is mass times acceleration: F = ma. Acceleration a is the rate of change of velocity: a = dv/dt. For a small interval of time

Δt, we can assume that $dv/dt \approx \Delta v/\Delta t$, where Δv is the amount by which v changes during the interval Δt. Consequently: $F \approx m\Delta v/\Delta t$, and therefore $F\Delta t \approx m\ \Delta v$.

When the racquet hits the ball, it squeezes the ball with the depth proportional to the speed of the racquet, which makes the ball deform. The impact force acting on the ball is proportional to the speed of racquet. Thus, the velocity change of the ball is proportional to the impact time (dwell time) T and the speed of racquet:

$$\Sigma(m\ \Delta V) = k\ v_{raq}\ T$$

Where Σ means the sum of the products of the mass and velocity of each point on the surface of the ball, m is the mass and ΔV is the velocity change of each point, V_{raq} is the speed of the racquet, T is the impact time, and k is an experimental coefficient.

3.2.1.5 Other Forces

Some forces are frequently neglected due to their negligible magnitudes such as air resistance force. The air resistance is a special type of frictional force that acts upon objects as they travel through the air. Like all frictional forces, the force of air resistance always opposes the motion of the object [6]. Air resistance force is proportional to the square of the speed times the frontal area of the object.

3.2.2 Newton's Laws of Motion

Newton's first and second laws of motion are applied in this project. The equation of Newton's laws: $a = \Sigma F / m$ is used in this project to calculate the displacement of the ball, where ΣF is the sum of all the forces acting on each point, and m is the mass of each point.

By definition, velocity is the integral of acceleration over time:

$$v = \int a dt \qquad (1)$$

If the acceleration is constant, we can evaluate the integral in (1), obtaining

$$v = a t + C \qquad (2)$$

Where C is the constant of integration. Assuming that $v = v_0$ when $t = 0$, (2) simplifies to

$$v = v_0 + a t \qquad (3)$$

However, in this application, the acceleration is in general not constant. We assume that it is approximately constant during a short interval Δt. Assume that the velocity is v at the beginning of the interval and $v + \Delta v$ at the end of it, (3) gives

$$v + \Delta v = v + a\Delta t \qquad (4)$$

Cancel v on both sides in (3), we get

$$\Delta v = a\Delta t \qquad (5)$$

By definition, displacement p is the integral of velocity over time:

$$p = \int v dt \qquad (6)$$

Using the same method we derived from (1) to (5), we get

$$\Delta p = v\Delta t \qquad (7)$$

3.3 Pressure

When a squash ball hits the racket strings and the wall and floor of the court, some of this energy is transformed into heat in the racquet strings, wall, floor, and surrounding air and some into sound. However, most of it becomes heat in the ball itself [10]. "One of the effects is: the air inside the ball, which was originally at normal atmospheric pressure, effectively becomes pressurized" [10]. The increase in pressure occurs during impact with the racquet, wall and floor. The effect of the pressure inside the ball at a point P is directed outwards, in the direction of the normal, and depends on volume:

$$F_{pressure} = K_p f(V) n$$

Where K_p is the constant of pressure, V is the volume, f (V) is a function (explained in 3.3.2) with V as input variable, and n is the normal at point P (figure 3-7).

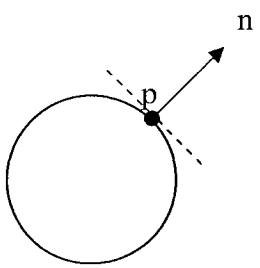


FIGURE 3-7 PRESSURE AT POINT P

3.3.1 Volume

Volume V equals 4/3 πr^3 when a squash ball is not only stable and but also spherical, where r is the nature radius of the ball. Once the ball deforms, the distances between points on the surface and the center of the ball are different so we need an approximate method to calculate the volume.

One approximate calculation is to divide the ball into thin disks of horizontal rings (as shown in figure 3-3); see the 2D orthogonal surface in figure 3-8.

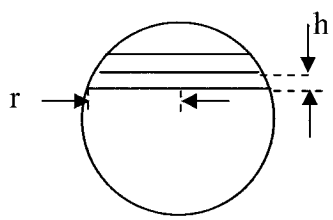


FIGURE 3-8 CONVEX SURFACE

The approximate volume is equal to the sum of the volumes of all the disks:

$$V = \Sigma (\pi r^2 h)$$

Due to the different speed of each point on the surface of the ball during the deformation, it may happen that the surface of the ball becomes concave at one of the poles, as shown in Figure 3-9, which shows a 2D projection of a ball that has become concave at one pole.

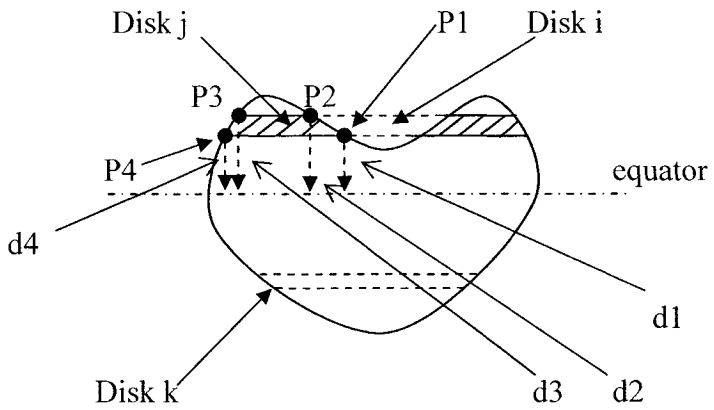


FIGURE 3-9 CONCAVE SURFACE

As shown in figure 3-9, there are 3 kinds of disks:

- Disk k: should be counted entirely.
- Disk j: should be counted except the middle disk i, which is the exterior of the ball.
- Disk i: should not be counted.

In the other words, disk j subtracting disk i forms an annulus (the patched part in figure 3-9), and it is the area of this annulus that we need.

So, how do we tell whether the disk should be added or subtracted in the volume calculation?

Let's start from the concave pole: we meet p1, p2, p3 and p4 in order, and we meet disk i before disk j. We should consider the two points on the same side of the disk in the concave orthogonal surface:

If the distance from Pi to the equator is shorter than the distance Pi+1 to the equator, this disk is outside the ball. As shown in figure 3-9, disk i (the patched part) is outside the ball because d1 <d2.

On the other hand, if the distance from Pi to the equator is longer than the distance Pi+1 to the equator, this disk is inside the ball. As shown in figure 3-9, disk j is inside the ball because d3 > d4 (d3 = d2, d4 = d1). We don't have to worry about the middle disk which is outside the ball because it is already subtracted as disk i before.

3.3.2 Pressure Calculation

When a gas expands or contracts slowly enough for its temperature to change, it obeys Boyle's Law, PV = constant. When the expansion or contraction is too fast to allow heat to enter or leave the system, the gas obeys the Adiabatic Gas Law (ADL), PV^{γ} = constant, where γ, the gas constant, depends on the gas.

The motion of a squash ball is very fast. Nevertheless, the ball does gain heat during repeated collisions, and its pressure goes up. On a millisecond timescale the compression of the air will be adiabatic, so the pressure increase can be calculated from the standard relation of ADL:

$$P V^{\gamma} = K \qquad (1)$$

Where K is a constant. If the pressure changes to $P + \Delta P$ and the volume changes to $V + \Delta V$, (1) gives:

$$(P + \Delta P)(V + \Delta V)^{\gamma} = K$$

Therefore:

$$(P + \Delta P)(V + \Delta V)^{\gamma} = PV^{\gamma} \qquad (2)$$

Dividing both sides by $(V+\Delta V)^{\gamma}$ in (2) gives:

$$P + \Delta P = P\, V^{\gamma} / (V+\Delta V)^{\gamma} \qquad (3)$$

Subtracting P from both sides in (3) gives:

$$\Delta P = P\, V^{\gamma} / (V+\Delta V)^{\gamma} - P \qquad (4)$$

Rewriting (4), we get

$$\Delta P = P\, (V^{\gamma} / (V+\Delta V)^{\gamma} - 1) \qquad (5)$$

Assume P_0 is the atmosphere pressure and V_0 is the original volume when the ball is stable before any impact, we get:

$$\Delta P = P_0\, (V_0^{\gamma} / (V_0+\Delta V)^{\gamma} - 1) \qquad (6)$$

Since $V_0+\Delta V$ is the changed volume of ball, let's say V', we get ΔP, the increased pressure inside the ball, which exerts force on the surface of the ball:

$$\Delta P = P_0\, (V_0^{\gamma} / V'^{\gamma} - 1) \qquad (7)$$

Rewrite (7), we get:

$$\Delta P = P_0\, ((V_0 / V')^{\gamma} - 1) \qquad (8)$$

For air, $\gamma \approx 1.4$.

3.4 One Point Simulation

In this section, we analyze a simulation of one point displacement, using all the forces discussed above: one patch of the spring mesh – one moveable point p and fixed neighbor points p1, p2, p3 and p4, as shown in figure 3-10.

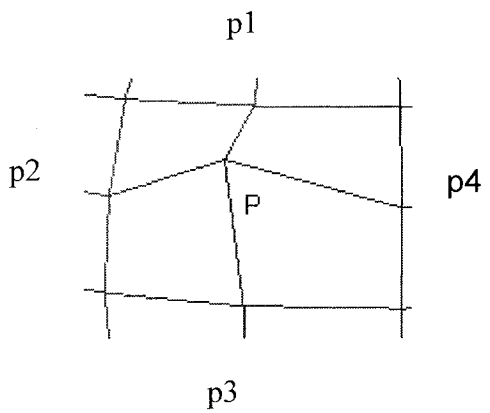


FIGURE 3-10 ONE POINT DISPLACEMENT

Here is what should happen for point p:

- Initially, point p is still because all the springs are at their natural lengths and there is no spring force.
- Displace point p front its initial position.
- Then calculate the spring force upon point p from four connecting neighbors p1, p2, p3 and p4, the acceleration, the velocity and the changing position of point p.
- If the simulation works, point p should move back to its equilibrium position, keep going to its maximum, then move back to its equilibrium position again.

- To stop the continuous circular motion, damping force is added on point p, so p will be slow down, the "circle" will get smaller, and p will stop at its initial position after a while.

This simulation was tested using C++, and it works well. Now we apply this simulation to the ball, allowing all the points on the surface of the ball to move, and adding the pressure force on all points.

Now to simulate the scenario of the ball hitting the wall:

- Put the wall at $x = 0$.
- Position the ball so that all points have positive x coordinates.
- Initialize the velocities of all points to (-v, 0, 0), so that the ball is moving towards the wall.
- In the computation, restrict x coordinates to positive values:

  If $x \mathrel{+}= v\,\Delta t$ would make $x < 0$, set $x = 0$.

  If $x = 0$, the point remains "stuck" to the wall.

  When $x > 0$, the point starts moving back.

4. Implementation

This project is implemented using C++ and OpenGL, with assistance from Concordia University Graphics Library (CUGL) [11] and GLUI User Interface Library [12] (described in section 4.3). CUGL provides a selection of class abstractions such as "point" and "vector" that speed up the development of this project.

There are two simulations implemented in this project

- The racquet hitting the ball and the ball rebounding from the front wall.
- The ball dropping and rebounding from the ground.

The initial scenarios of the two simulations are almost identical, as shown in figure 4-1. The difference is only that there is no racquet in the ball dropping simulation.

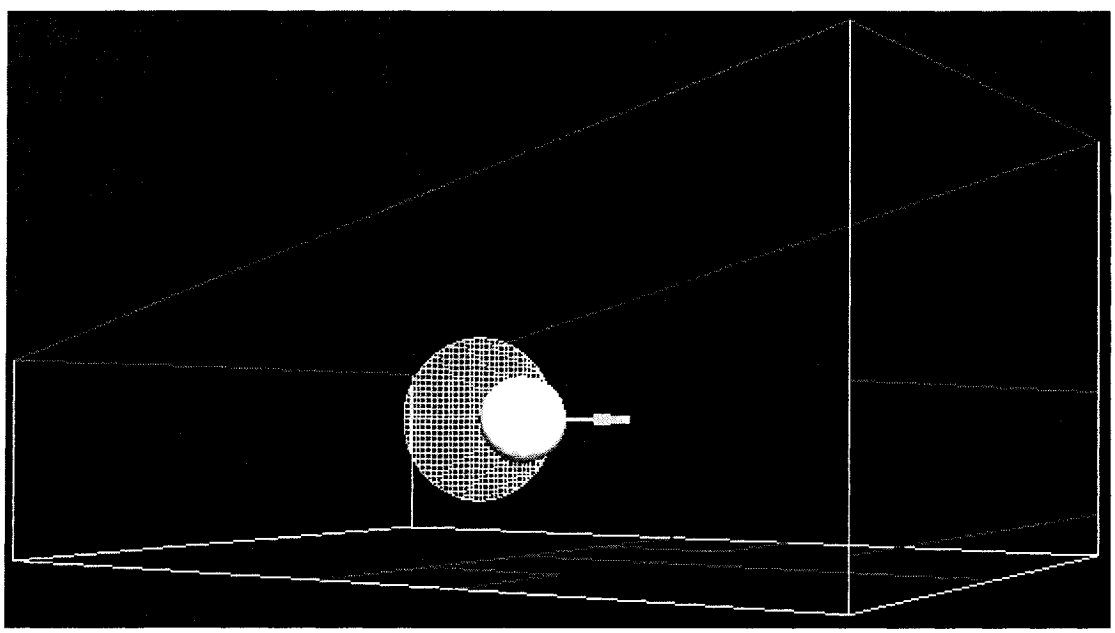


FIGURE 4-1 MAIN WINDOW

4.1 Class Definitions

There are four main classes to describe 3D objects in a squash court: ball, spring, racquet, and court.

Ball Class - the ball class has private shape, position and dynamic attributes, including:

- Number of stacks and number of slices.
- 2-D array of areas of all trapezoids on the surface.
- 2-D array of the combination forces of all points on the surface.
- 2-D array of springs (explained in 3.5.3) connecting to each point.
- 2-D array of accelerations of all points.
- 2-D array of velocities of all points.
- 2-D array of displacements of all points.

The ball class has public functions that are called in OpenGL functions, environment setup functions, and user-control functions, including:

- Functions to initialize attributes of the ball.
- A function to calculate the changing volume of the ball.
- Functions to update the dynamic attributes of points on each unit time.
- A function to compute normals of points.
- A function to render the ball.

Racquet Class - as a 3D object, racquet class has shape, position and dynamic attributes similar to the ball class. The racquet is assumed to be solid in this project so the racquet does not contain any spring structure.

Spring Class - the spring Class consists of natural lengths of four connecting springs of a mass point. Natural lengths are compared with actual lengths of the springs at each time unit to get spring forces.

Court Class - the court class has measurements of a standard squash court. They are used to render a squash court. In addition, they are the boundaries for a moving squash ball.

4.2 Position Ball and Racquet

In order to show the ball close to the center of the viewport initially, the ball is positioned right on top of the half court line and the short line of the court (the red spot in figure 4-2), and the height to the ground is 1.5m, about 3 quarters of the height of the back wall.

Since the distance from the racquet to the ball does not influence the movement and deformation of the ball, the racquet is positioned right behind the ball (21.6mm from the center of the ball) as shown in figure 4-3.

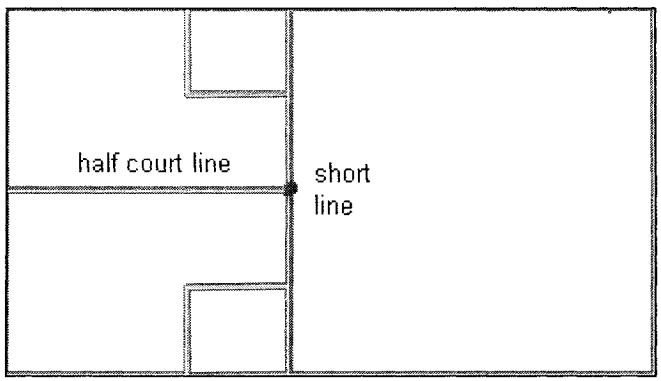


FIGURE 4-2 SQUASH COURT AT TOP VIEW

4.3 GLUI Controls

GLUI is a GLUT-based C++ user interface library that provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window-system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management [12].

In this project, there are two GLUI windows - bottom window and side window within the main window, as shown in figure 4-3.

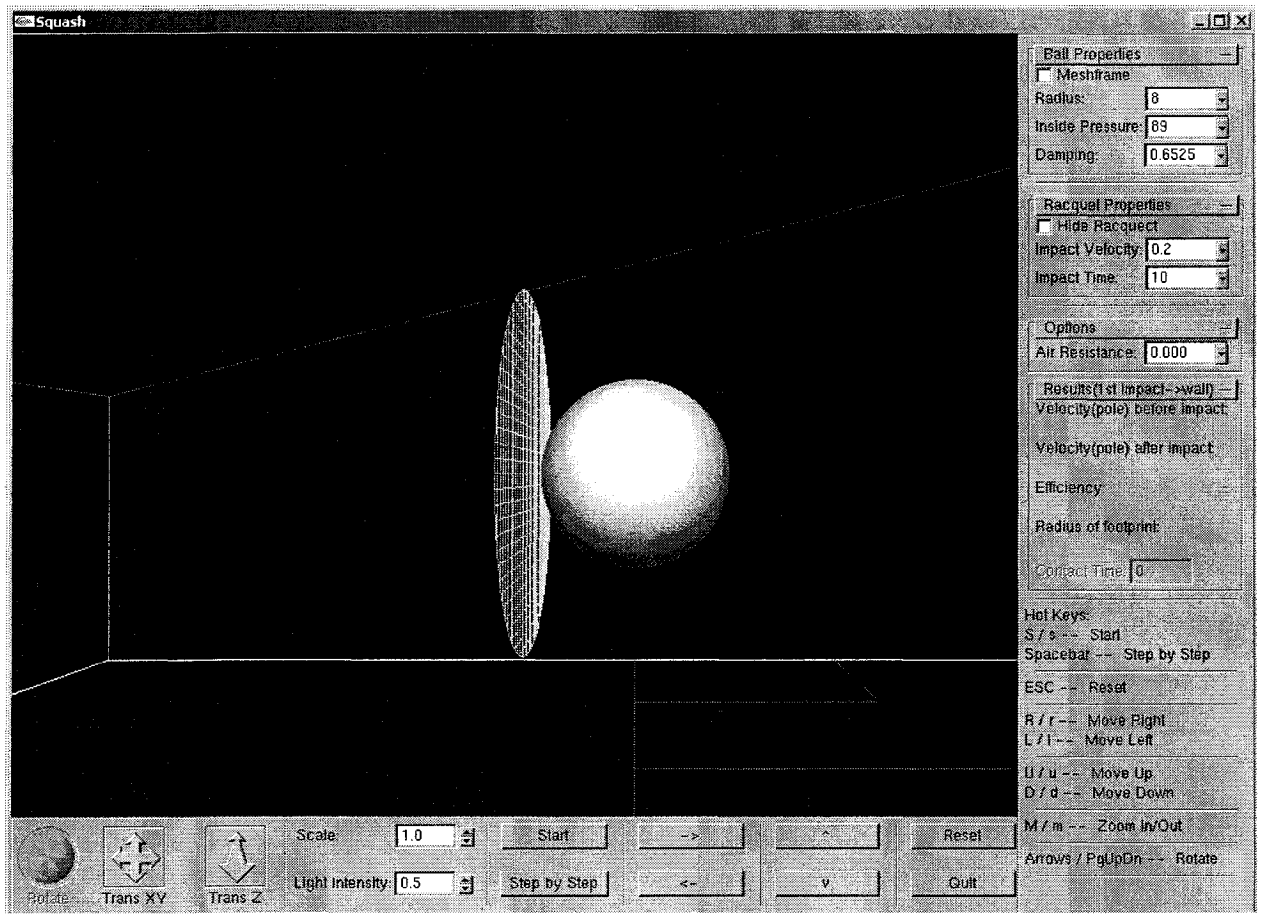


FIGURE 4-3 GLUI WINDOWS WITHIN MAIN WINDOW

4.3.1 Side Window

The side GLUI window provides users with objects' properties and options, real-time results and keyboard controls.

4.3.1.1 Ball Properties

The ball can be rendered in two methods – either by filling or by mesh frame (figure 4-4), chosen by checking the checkbox "Meshframe" (figure 4-5).

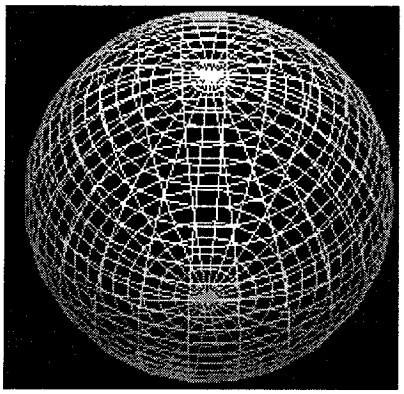


FIGURE 4-4 BALL MESH FRAME

Users can see the differences in the deformation and the movement of the ball by choosing different radius, inside pressure and damping coefficient (figure 4-5).

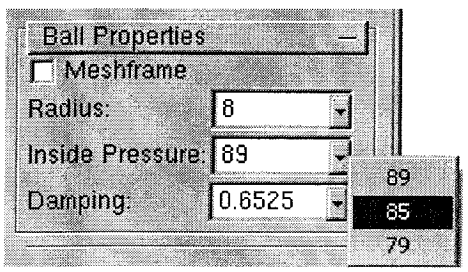


FIGURE 4-5 BALL PROPERTIES

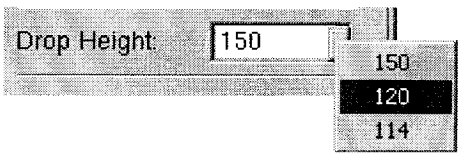


FIGURE 4-6 HEIGHT CONTROL

In the ball dropping simulation, there is a spinner control "Drop Height" (figure 4-6) right below the ball properties control. The higher the ball is dropped from, the

higher the ball bounces and more the ball deforms. Real-time results output gives the details of relevant data (described in 4.3.1.3).

4.3.1.2 Racquet Properties

In the racquet hitting the ball simulation, as mentioned in 3.2.1.4, the speed and impact time of the racquet determines the velocity of the ball. Users can see how the combination of velocity and dwell time (figure 4-7) acts on the ball. Users might like to move away the racquet to focus on the ball movement chosen by checking the checkbox "Hide Racquet".

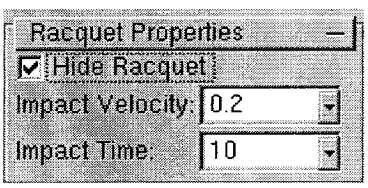


FIGURE 4-7 RACQUET PROPERTIES

4.3.1.3 Real-time Results output

The results output is based on the online Java applet that simulates the bounce of an air-filled ball [14]. Figure 4-8 is the results output of the simulation that the ball is hit by the racquet and bounces back from the front wall.

The top two values are the velocities of the ball before and after bouncing from the front wall and the efficiency is the quotient of these two values. The footprint is the maximum radius of the contact area. The contact time is counted from the moment when the ball hits the front wall to the moment when it leaves the wall.

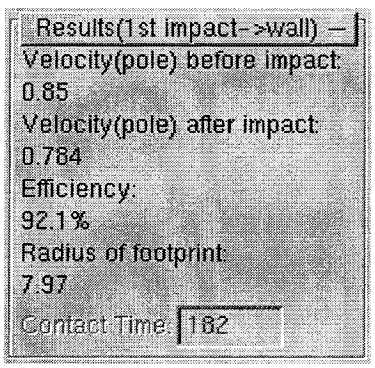


FIGURE 4-8 RESULTS OUTPUT

4.3.1.4 Keyboard controls

For convenience, a group of hot keys is used to operate the applications, which are defined by GLUT callback function glutKeyboardFunc(). We show hotkey usage at the bottom of the side GLUI window (figure 4-3) instead of using standard C++ function "cout " to output it, so users can easily notice it.

- S / s - start the simulations
- Spacebar - step by step control of the simulations
- ESC - reset the simulations
- R / r- move the scene to the right

- L/ l - move the scene to the left
- U / u - move the scene upwards
- D/ d - move the scene downwards
- M / m - zoom in or zoom out the scene
- Left/ Right Arrows - rotate the scene around Z-axis
- Up/ Down Arrows - rotate the scene around X-axis
- PgUpDn - rotate the scene around Y-axis

4.3.2 Bottom Window

The bottom GLUI window provides users with rotation and translation controls, scale control, light intensity control and some buttons.

4.3.2.1 Rotation and Translation

The rotation control displays as a checkboard-texture sphere (figure 4-9), which users can manipulate directly. The current rotation of the control is passed to the application as an array of 16 floats, representing a 4x4 rotation matrix. This array can be passed to OpenGL directly to rotate an object, using the function glMultMatrix() [12].

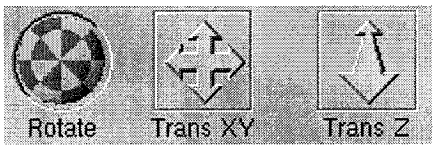


FIGURE 4-9 ROTATION AND TRANSLATION CONTROLS

Two types of translation controls - XY and Z translation controls and used in this project (figure 4-9). Translation controls allow users to manipulate X, Y, and Z values for the entire or part of scene chosen by clicking on and dragging on-screen arrows.

4.3.2.2 Scale

Compared with the hotkey 'M' and 'm', Scale spinner provides continuous cycling over a range of float-point values (figure 4-10), which are the inputs of function glScaled ().

FIGURE 4-10 SCALE SPINNER

4.3.2.3 Lighting

It is hard to create the illusion of the real world without lighting. The OpenGL lighting model consists of four components: ambient, diffuse, specular, and emitted. These four components can work individually or jointly. A squash court needs to be

lit by some scattered lights. There are two lights - light0 and light1 "installed" on top on the court. They are ambient and diffuse light sources:

```
// Initialize the lights
GlEnable (GL_LIGHTING);
GlEnable (GL_LIGHT0);
GlLightfv (GL_LIGHT0, GL_POSITION, light0_pos);
GlLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
GlLightfv (GL_LIGHT0, GL_AMBIENT, light_ambient);

GlEnable (GL_LIGHT1);
GlLightfv (GL_LIGHT1, GL_POSITION, light1_pos);
GlLightfv (GL_LIGHT1, GL_DIFFUSE, light_diffuse);
GlLightfv (GL_LIGHT1, GL_AMBIENT, light_ambient);
```

OpenGL light sources can be turned on and off. Users can adjust the intensity of the light (figure 4-11) gradually to animate the scene.

FIGURE 4-11 LIGHT INTENSITY SPINNER

From left to right, figure 4-12 shows the squash ball illuminated by the lights with ambient value 0, 0.5 and 1.

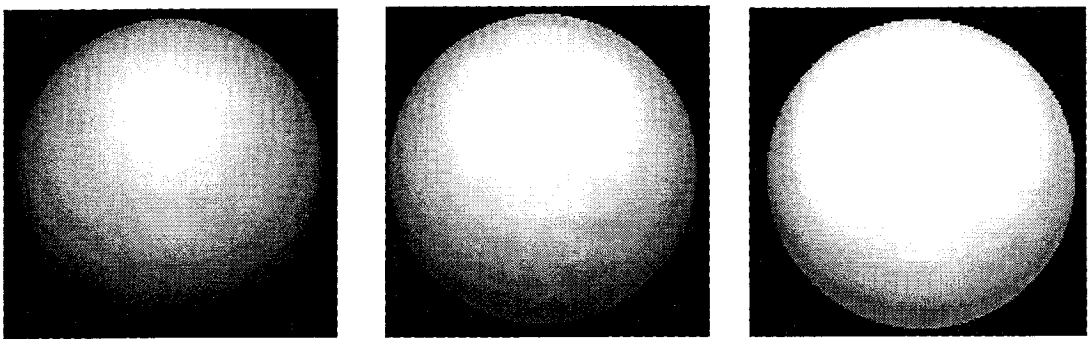


FIGURE 4-12 AMBIENT LIGHT

As mentioned in 3.1.2, OpenGL needs to know the direction of the normals to perform lighting calculations at each point of the surface that it is rendering. Figure 4-13 shows the ball illuminated without telling the normals and figure 4-14 shows the ball illuminated with the normals.

FIGURE 4-13 LIGHTING EFFECT WITHOUT NORMALS

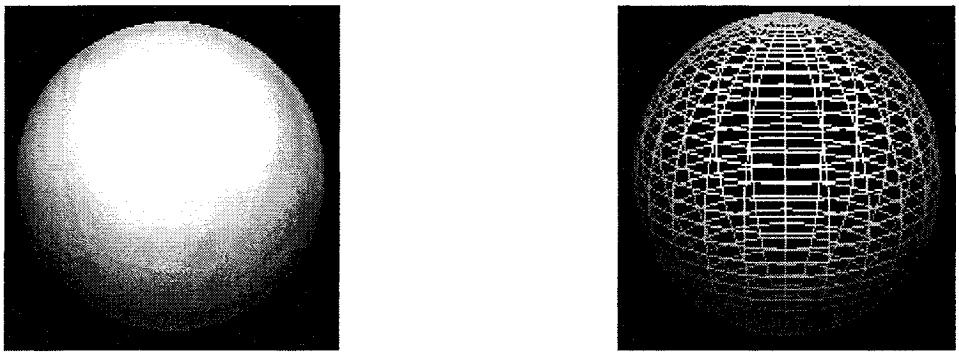


FIGURE 4-14 LIGHTING EFFECT WITH NORMALS

4.3.2.4 Buttons

As shown in figure 4-15, the first seven buttons issues the same actions as the relative hotkeys, and the last button "quit" ends the applications.

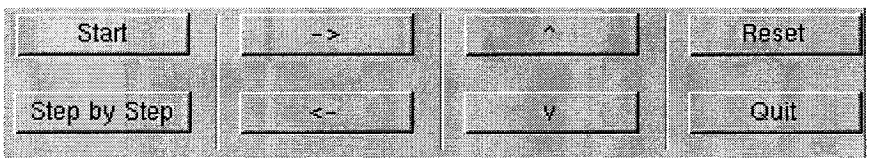


FIGURE 4-15 THE GROUP OF BUTTONS

4.3.3 Window Management

GLUI provides a friendly user interface enable animated graphics with user interaction.

4.3.3.1 Live Variables

GLUI associates live variables with most types of controls. Live variables are regular C variables that are automatically updated whenever the user interacts with a GLUI control. These variables make the GLUI interface transparent to the application. For example:

checkbox1 = glui-> add_checkbox_to_panel ball_panel, "Meshframe", filledFlag);

The Boolean variable filledFlag will be automatically toggled between one and zero whenever the user checks or unchecks the control. In result, the ball will be rendered filled or meshed according the updated filledFlag in the GLUT display callback function.

4.3.3.2 Callbacks

GLUI controls can generate callbacks when their values change. For example:

//Create listbox of pressure

GLUI_Listbox *list1 = glui->add_listbox_to_panel (ball_panel, "Inside Pressure:", &curr_pressure_idx, PRESSURE_ID, control_cb);

Control_cb (int control_id) is a callback function that will be called when the value of curr_pressure_idx is changed, and the control ID PRESSURE_ID will be passed the callback function as an input.

4.3.3.3 Synchronization

One new feature in GLUI version 2 is function sync_live_all() that automatically synchronizes all live variables in all GLUI windows simultaneously. This function is evoked in the GLUT idle callback function.

4.4 Problems and Solutions

Some problems and solutions during the implementation are described in this section.

4.4.1 Boundaries Check

The calculation of forces and displacements starts from the pole which first impacts the racquet, the wall, and the floor because it is the first part to get distorted. As shown in figure 4-16, the procedure of the simulation is:

- The racquet impacts pole1 and the ball moves forwards.
- Pole2 hits the front wall and the ball bounces back.
- Pole1 hits the back wall and the ball bounces back.

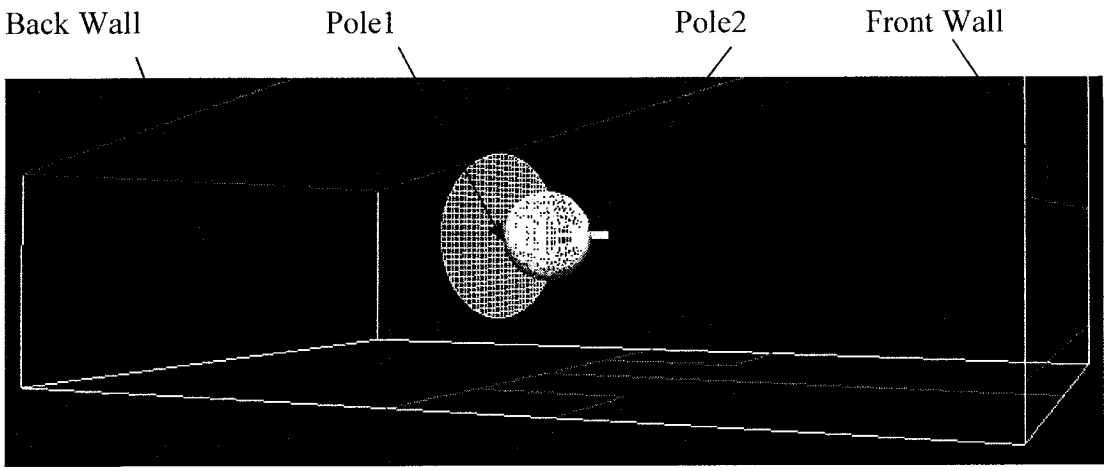


FIGURE 4-16 TWO POLES OF THE SQUASH BALL

To distinguish the direction of the ball movement, some flags are set and reset by checking boundaries, for instance: when the ball hits the front wall, the "hitFlag" that indicates the ball and the wall impact will be set.

```
//Check the front wall boundary
if(tmpArr[0][0][0] >court.frontWallVer[0][0])
{
        impactFlag =false;
        hitFlag = true;
        …
}
```

4.4.2 Conflicts of Controls

Some GLUI controls cannot be touched if some other controls are activated. For example, if users click the "start" button to start a continuous simulation. The "step by step" button should not be touched until the simulation finishes, and vise versa. Therefore, to prevent a conflict, one button will be disabled when the other is activated by using function GLUI_Control::disable(void).

Another example: the physical features of the ball should not allow to be changed once the ball moves, whereas the ball can still be rendered either solid or frame. As shown in figure 4-17, the radius, the pressure and the damping listboxes are disabled.

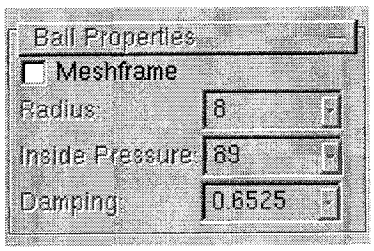


FIGURE 4-17 ENABLE AND DISABLE CONTROLS

4.4.3 Rendering Racquet

The left picture in figure 4-18 is a standard squash racquet. The frame of the racquet is not a center-symmetric oval and the vertical strings of the racquet are not parallel.

Since the racquet is assumed solid in this project, the racquet is simply rendered as the right picture in figure 4-18.

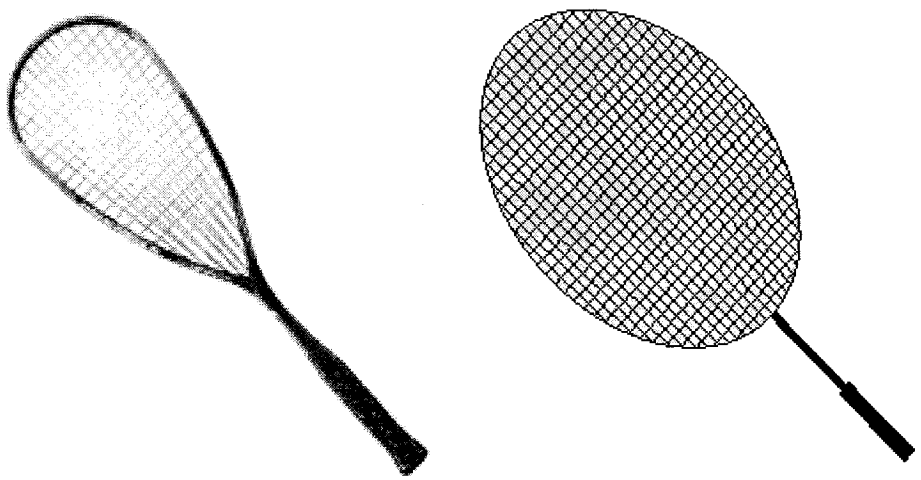


FIGURE 4-18 SQUASH RACQUET

4.5 Sample Code

In this section, we take the ball dropping on the ground as an example. We explain one iteration in details, from forces to position calculations on vertex vec[i][j], where i is the index of stacks and j is the index of slices.

```
//calculate actual lengths, velocity differences, and compress or stretch lengths
//between vec[i][j] and its four neighbors.
        actlen1 = vec[i][j]-vec[i][j-1];
        veloff1 = vel[i][j]-vel[i][j-1];
        actoff1 = spr[i][j].natlen1 -  actlen1.length();
```

```
        actlen2 = vec[i][j]-vec[i+1][j];
        veloff2 = vel[i][j]-vel[i+1][j];
        actoff2 = spr[i][j].natlen2 -  actlen2.length();

        actlen3 = vec[i][j]-vec[i][j+1];
        veloff3 = vel[i][j]-vel[i][j+1];
        actoff3 = spr[i][j].natlen3 -  actlen3.length();

        actlen4 = vec[i][j]-vec[i-1][j];
        veloff4 = vel[i][j]-vel[i-1][j];
        actoff4 = spr[i][j].natlen4 -  actlen4.length();

//calculate the spring forces on vec[i][j]
        f = fk *((actoff1 * actlen1.unit() +actoff2 * actlen2.unit()
                        +actoff3 * actlen3.unit() +actoff4 * actlen4.unit()));

//add damping force on vec[i][j]
        f - = damping *(veloff1 + veloff2 + veloff3 + veloff4);

//add inside pressure if the volume excess is negative
        if (curVol<defVol)
                f += curPressure* area[i]*normal[i][j];
```

```
//get the acceleration of vec[i][j], including gavity and air resistance accelerations
        acc[i][j] = f  / massVec +grav-fair *vel[i][j].unit()* vel[i][j].length()*
                vel[i][j].length()* area[i] );

//get the  velocity of vec[i][j]
        tmpVel[i][j] =  area[i] *dt;

//get the position of vec[i][j]
        tmpArr[i][j] += tmpVel[i][j] *dt;

//if vec[i][j] hits the ground, restrict y value of vec[i][j] to y value of the ground
        if (tmpArr[i][j][1] < court.floorVer[0][1]){
                tmpArr[i][j][1] = court.floorVer[0][1];

//record the maximum footprint
                if (max_footprint< fabs(tmpArr[i][j][0]))
                        max_footprint = fabs(tmpArr[i][j][0]);}

//if vec[i][j] hits the ground, restrict the velocity on y direction of vec[i][j] to zero
        if (tmpArr[i][j][1] = = court.floorVer[0][1])
                tmpVel[i][j][1]= 0.0;
```

5. Results

In this chapter, we show some experimental results with a series of screen shots. We explain how physical properties affect the motion of the ball according to the output results. The following table lists the physical parameters we used:

Parameters	Ball/Racquet impact	Ball Dropping
spring constant	0.96	0.96
pressure constant	89	79
damping constant	0.6525	0.6525
racquet speed	0.2	not applicable
impact time	10	not applicable
gravity acceleration	not applicable	-0.0098

5.1 Simulation One - Ball /Racquet and Ball/Wall Impact

This section illustrates the two scenarios separately - the racquet hitting the ball and the ball hitting the front wall.

5.1.1 Screen Shots

The following two groups of screen shots, which are the racquet impacting the ball and the ball hitting the wall are taken under the default setting of the physical environment as shown in figure 4-3 using "StepbyStep" button.

Figure 5-1 (figure 5.1.1 - 5.1.15) shows fifteen typical screen shots taken during the sequence of the squash ball being impacted by the racquet, deforming, and moving forwards.

- Figure 5.1.1 - 5.1.3:

  The left pole squeezes in towards the center of the ball due to the impact force.

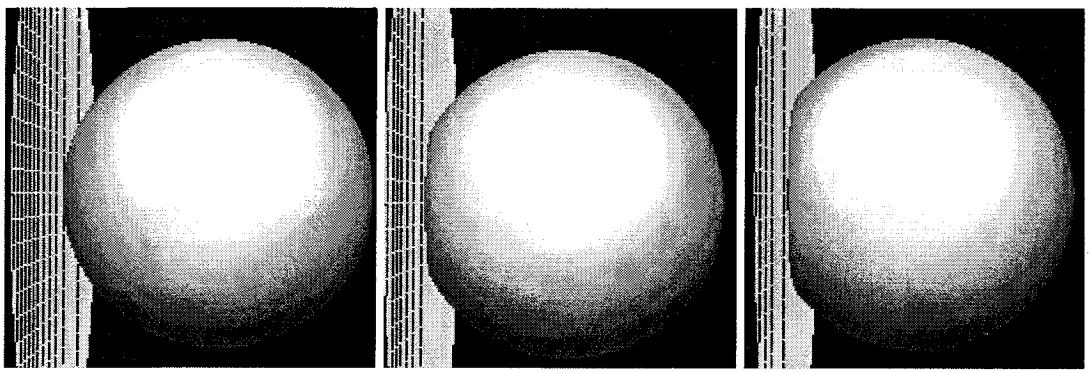


Figure 5.1.1 Figure 5.1.2 Figure 5.1.3

- Figure 5.1.4 - 5.1.6:

  The ball starts moving forward due to forces generated by the contracted spring and to increasing pressure caused by decreasing volume.

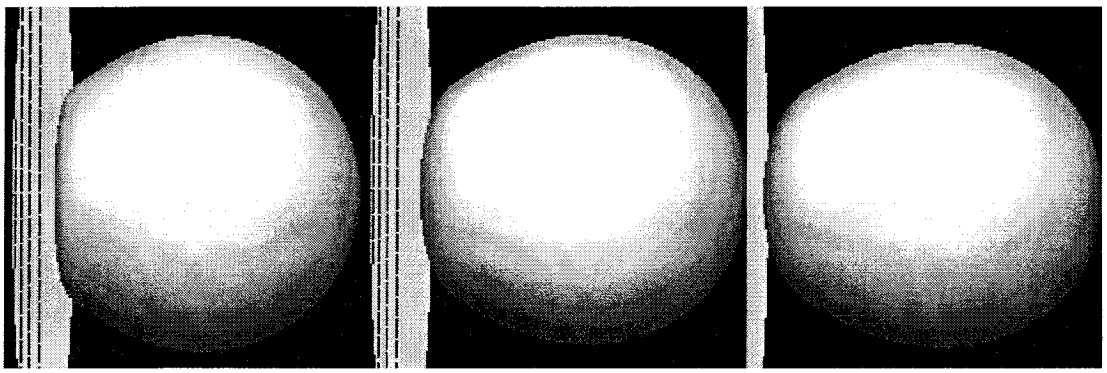


Figure 5.1.4 Figure 5.1.5 Figure 5.1.6

- Figure 5.1.7 -5.1.8:

  The ball leaves the racquet. The springs are stretched so the poles are distorted away from the centre of the ball.

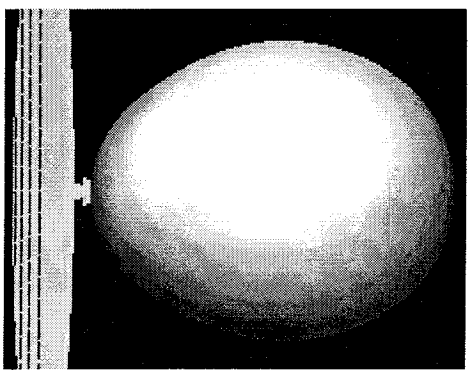


Figure 5.1.7

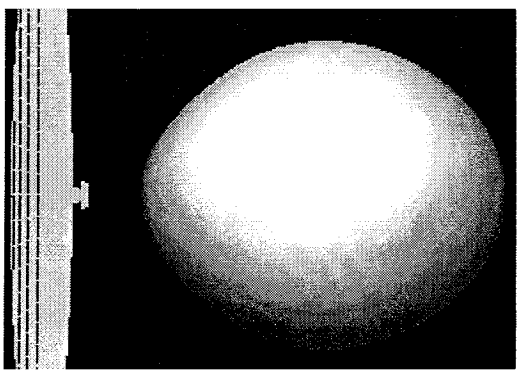


Figure 5.1.8

- Figure 5.1.9 - 5.1.12:

  The ball keeps going. The springs pass their equilibrium position and start to compress.

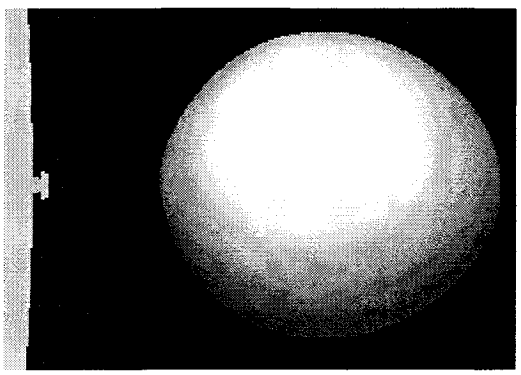


Figure 5.1.9

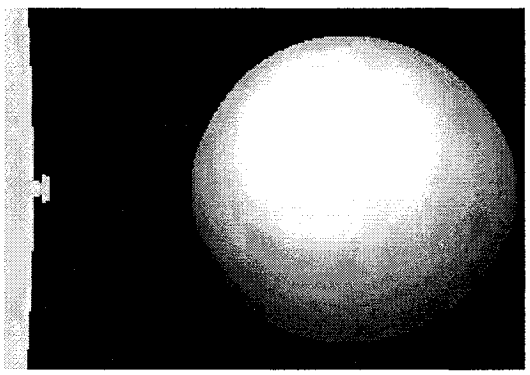


Figure 5.1.10

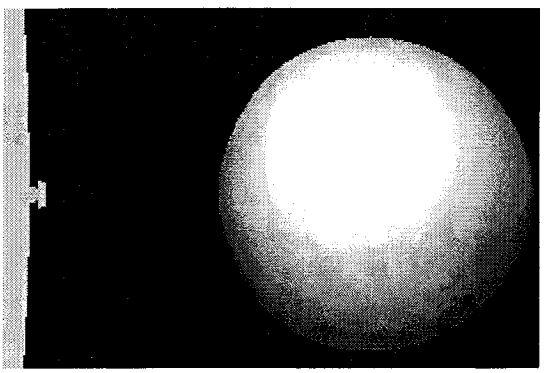


Figure 5.1.11

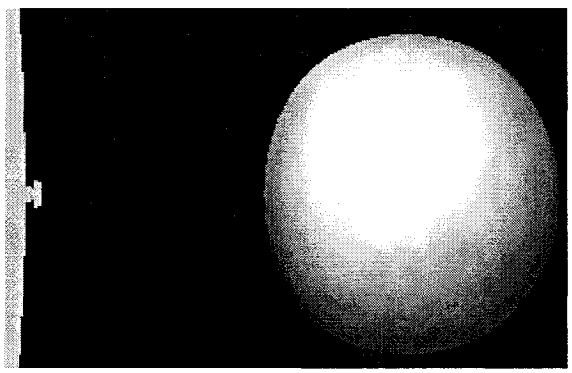


Figure 5.1.12

- Figure 5.1.13 - 5.1.15:

  The springs relax to its equilibrium position.

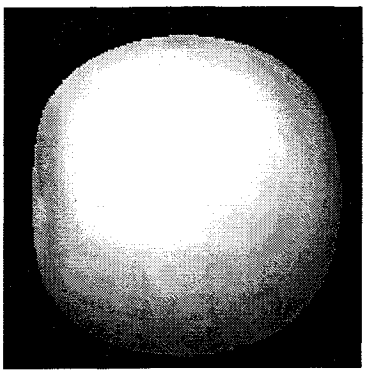


Figure 5.1.13

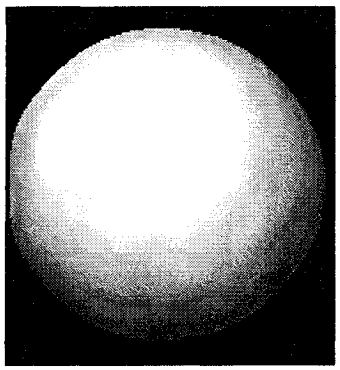


Figure 5.1.14

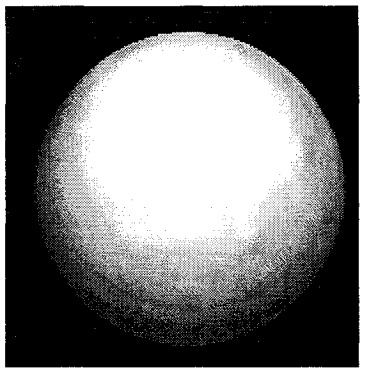


Figure 5.1.15

FIGURE 5-1 RACQUET IMPACTING BALL

Figure 5-2 (figure 5.2.1 - 5.2.13) shows the process of the squash ball hitting the wall then bouncing back. The process of the deformation is similar to the ball impacting the racquet. For an ideal visual effect, we use "Right" button and "TranXY" control to move the front wall close to the middle of the view and make the two vertical edges "stick" together from users' view. See the white line in figure 5.2.1.

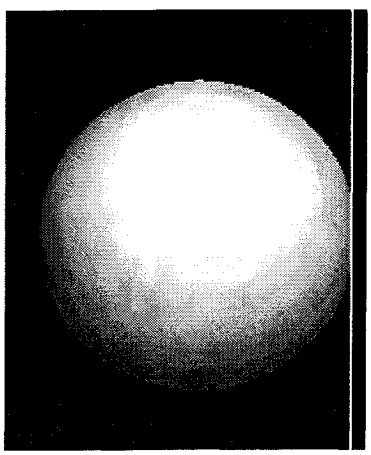


Figure 5.2.1

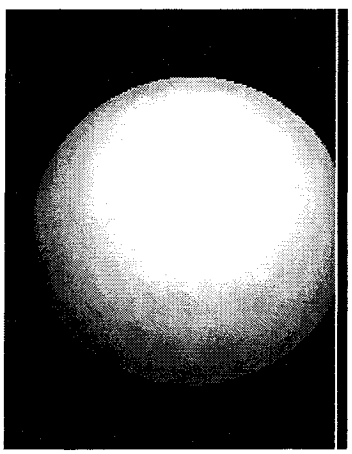


Figure 5.2.2

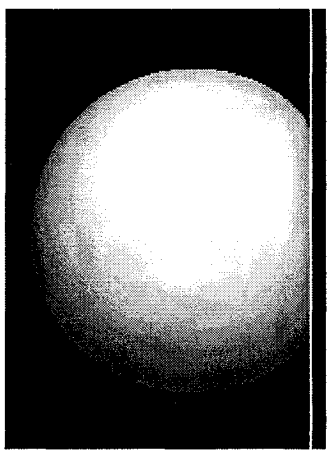


Figure 5.2.3

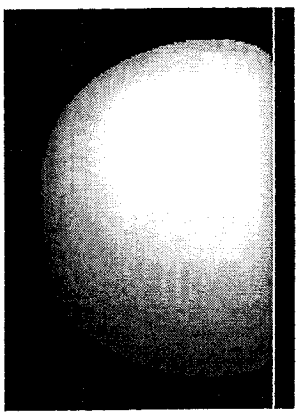


Figure 5.2.4

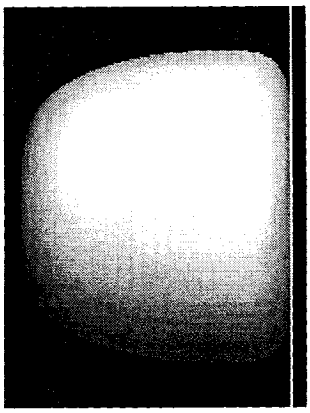


Figure 5.2.5

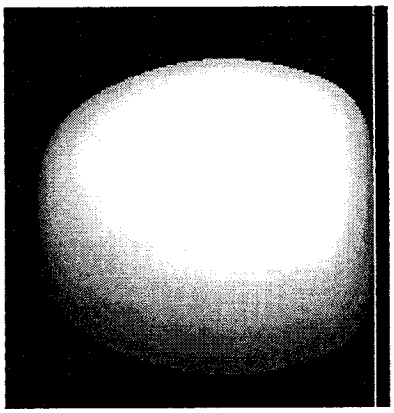


Figure 5.2.6

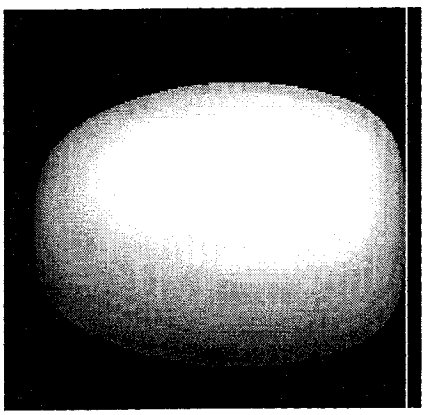


Figure 5.2.7

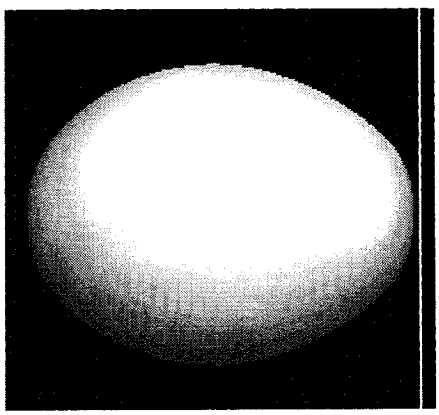


Figure 5.2.8

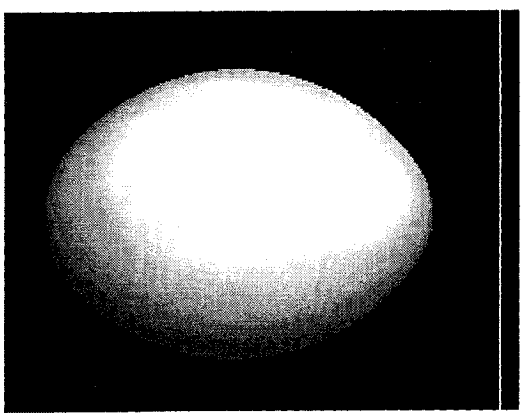


Figure 5.2.9

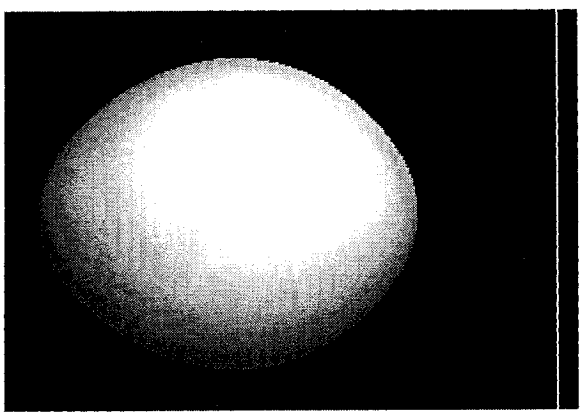


Figure 5.2.10

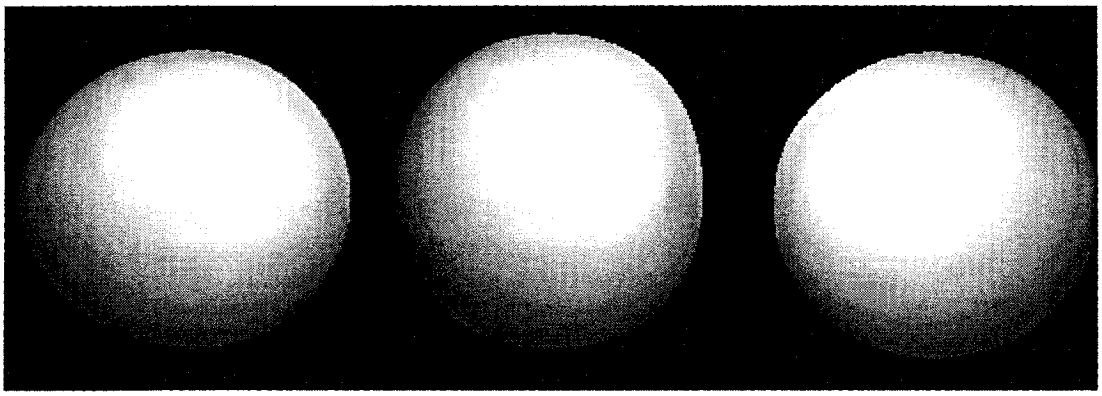


Figure 5.2.11 Figure 5.2.12 Figure 5.2.13

FIGURE 5-2 BALL HITTING WALL

5.1.2 Physical Environment Controls

We used a scenario in simulation one - the ball hitting the front wall and bouncing back and the result output in the right GLUI window to analyze the physical properties. The meanings of the data in output have been already described in section 4.3.1.3. We picked the velocity of the right pole to present the velocity of the ball:

Velocity (pole) before impact:

Recorded at the moment when the right pole impacts the front wall. Since the ball has traveled half of the court after served by the racquet, the ball is almost in its equilibrium form (figure 5.2.1) and the velocity of the right pole can represent the velocity of the whole ball.

Velocity after (pole) impact:

Recorded at the moment when the left pole hits the back wall. After moving ball from the front wall, the ball is almost in its equilibrium form so we still use the right pole to sample the velocity of the ball.

5.1.2.1 Pressure

Figure 5-3 shows three results, with the corresponding pressure control above. All the other controls are the default values. We can see that the bigger the inside pressure constant is, the less kinematic energy (velocity) is dissipated during the deformation.

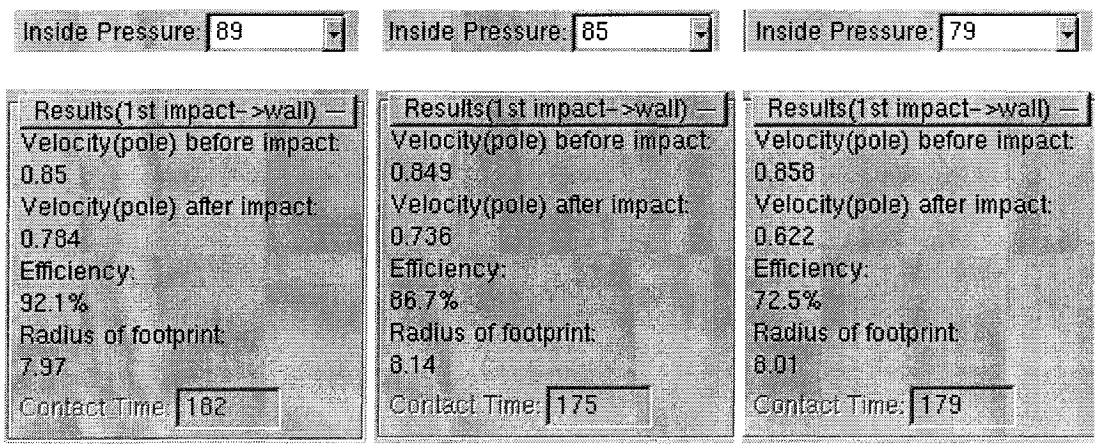
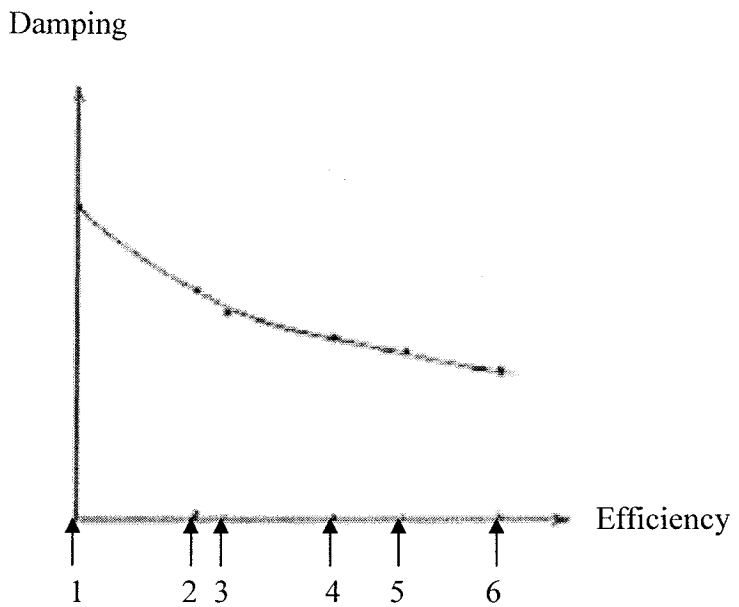


FIGURE 5-3 RESULTS UNDER DIFFERENT PRESSURES

5.1.2.2 Damping

The property of damping enables an object to dissipate energy, usually by conversion of kinetic energy into heat energy. Damping consumes mechanical energy from the system and attenuates vibrations more quickly. As shown in Figure 5-4, a small difference in damping makes a large difference in efficiency, which is the quotient of the velocities of the ball before and after bouncing from the front wall.



	Damping	Efficiency
1:	0.6525	92.1%
2:	0.6534	68.2%
3:	0.6536	60.8%
4:	0.6544	55.1%
5:	0.6549	49.6%
6:	0.6556	45.1%

FIGURE 5-4 RESULTS WITH DIFFERENT DAMPING

5.1.2.3 Impact Velocity and Dwell Time

As mentioned in 3.1.2.4, the velocity of the ball increases when the impact time and the speed of the racquet increase. Figure 5-5 and figure 5-6 show the results when changing dwell time or speed of the racquet. All the other physical properties remain default values.

The velocity of the right pole of the ball when leaving the racquet is output in the result panel.

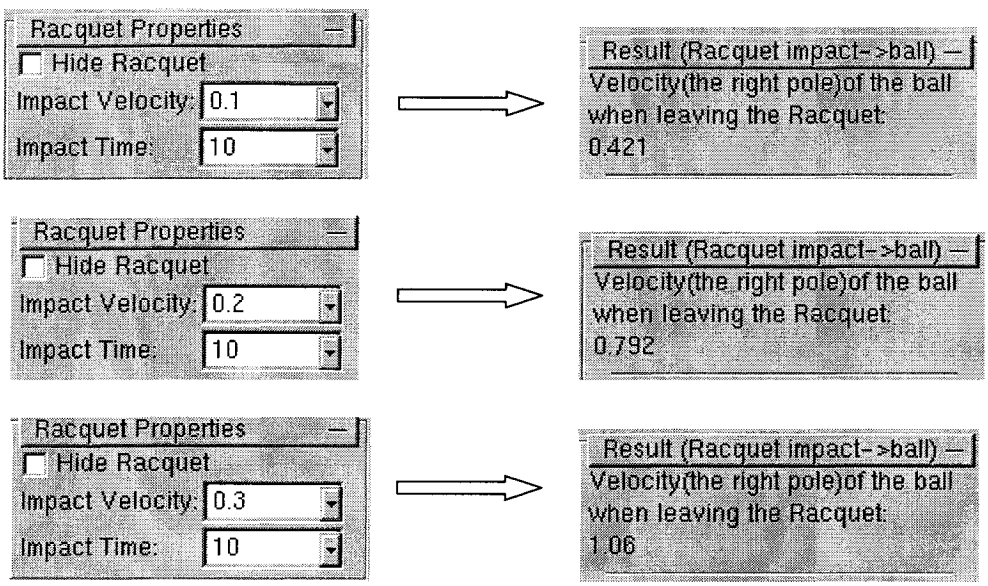


FIGURE 5-5 RESULT WITH DIFFERENT RACQUET SPEED

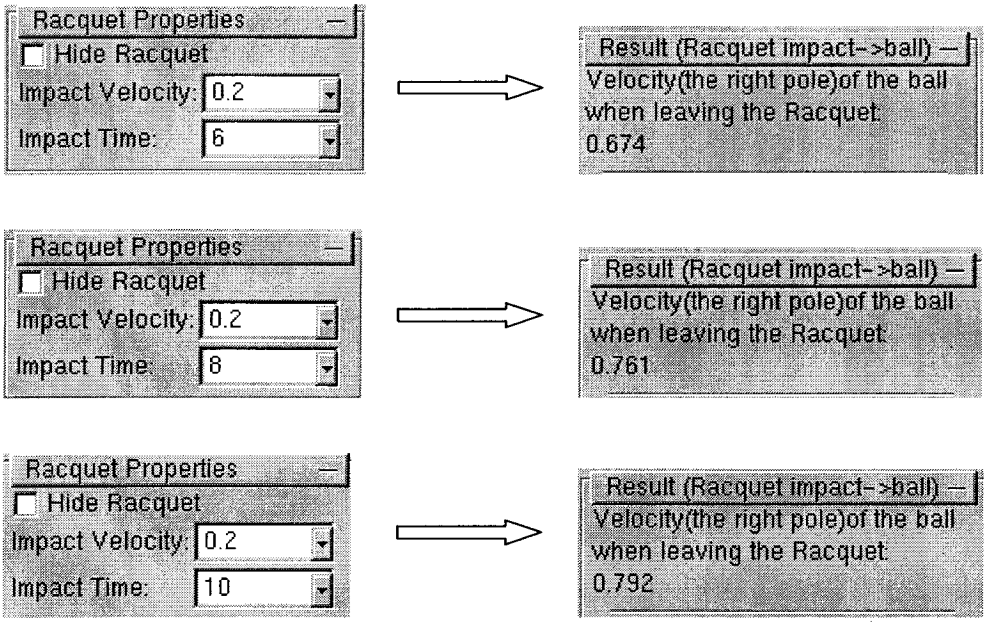


FIGURE 5-6 RESULT WITH DIFFERENT RACQUET DWELL TIME

5.2 Simulation Two - Ball/Floor Impact

The initial picture of the ball dropping simulation (figure 5-7) is similar to the one of the racquet/ball/wall simulation. The results output is similar to the output in the 2D Java applet [14]. In the ball dropping simulation, the efficiency is the height of the bounce compared to the drop height.

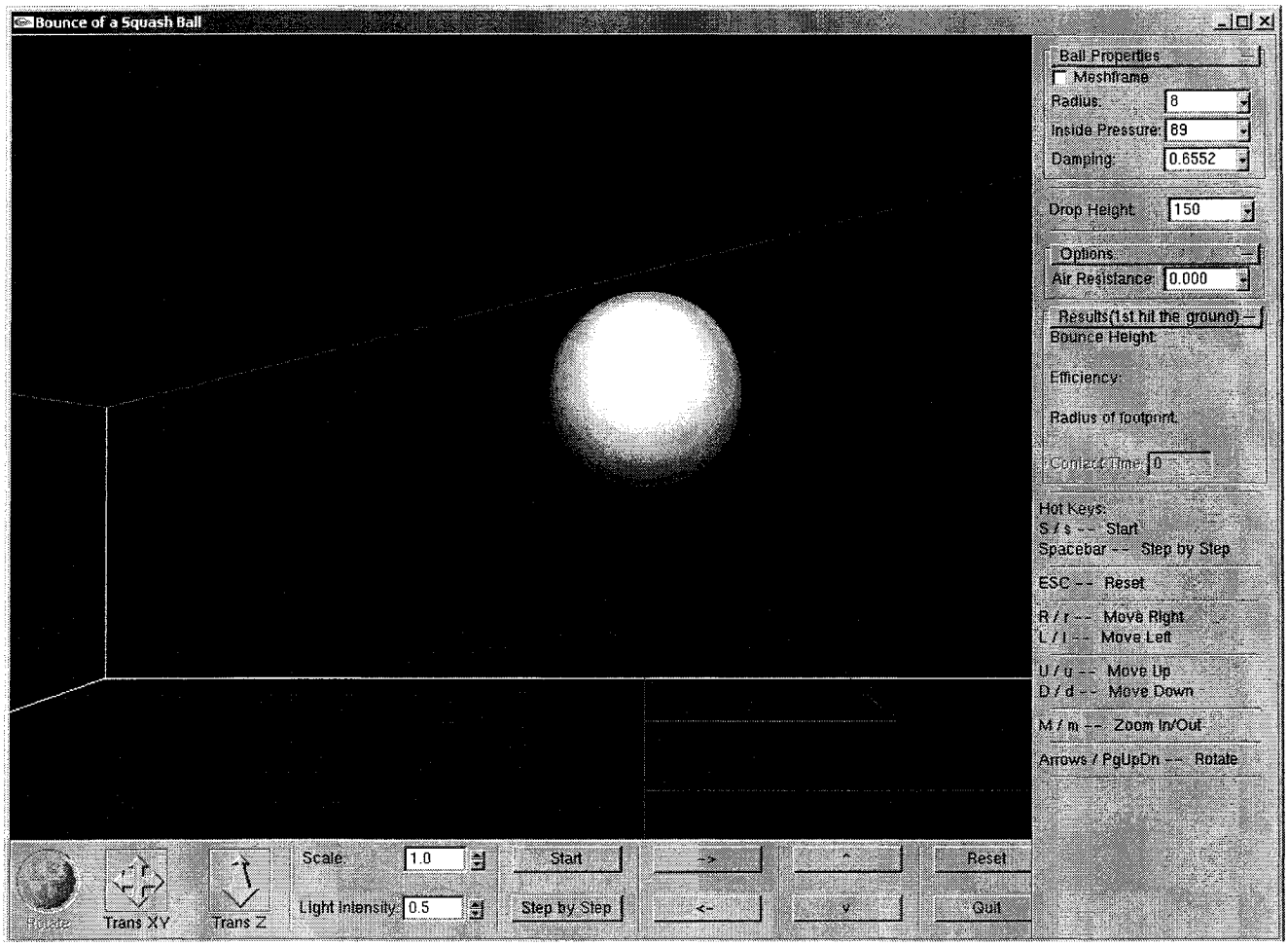


FIGURE 5-7 INITIAL WINDOW OF BALL DROPPING SIMULATION

5.2.1 Screen Shots

Figure 5-8 shows the process of the squash ball hitting the floor and rebounding upwards. We use "TranXY" control to adjust the view, make the front and back edges of the floor "stick" together from users' view.

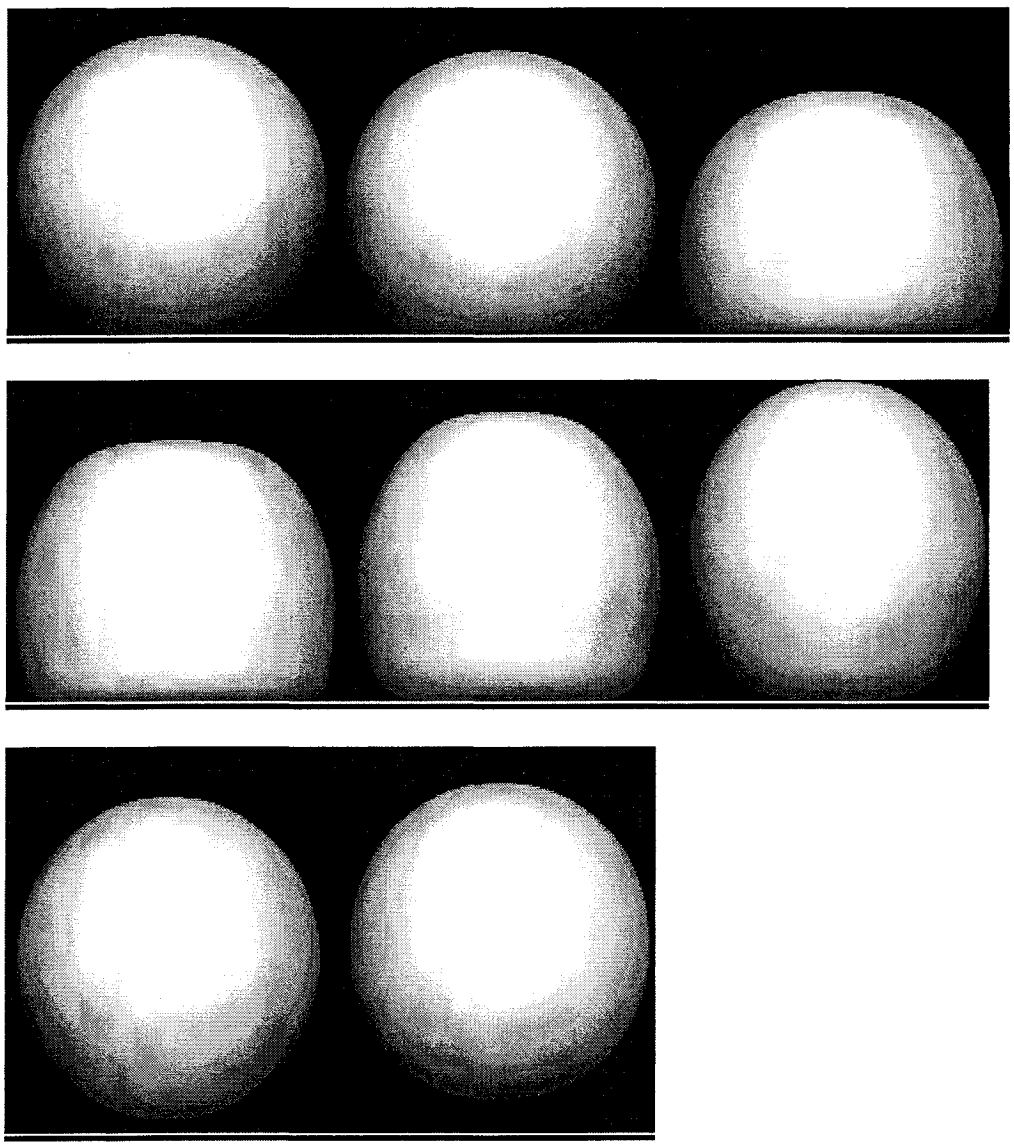
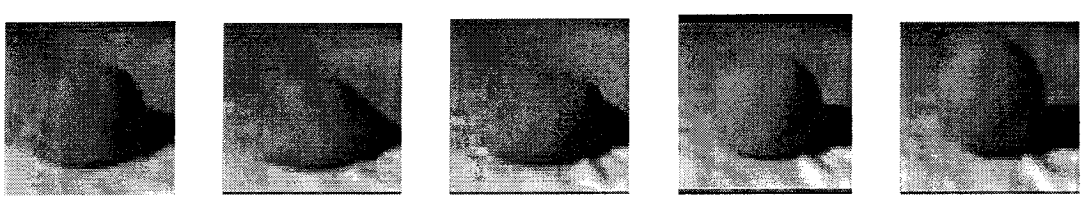


FIGURE 5-8 BALL HITTING FLOOR

Compare the following video clips we show in section 2.1.3, our simulation is close to the real ball bounce. Our ball squeezes inwards less when hitting the ground.



5.2.2 Drop Height

The "efficiency" in results output means the height of the bounce compared to the drop height. We drop the ball from different heights to test this efficiency. The results show that efficiency is proportional to the drop height, as shown is figure 5-9. All the settings in these tests were the default values except the pressure for which we chose 79.

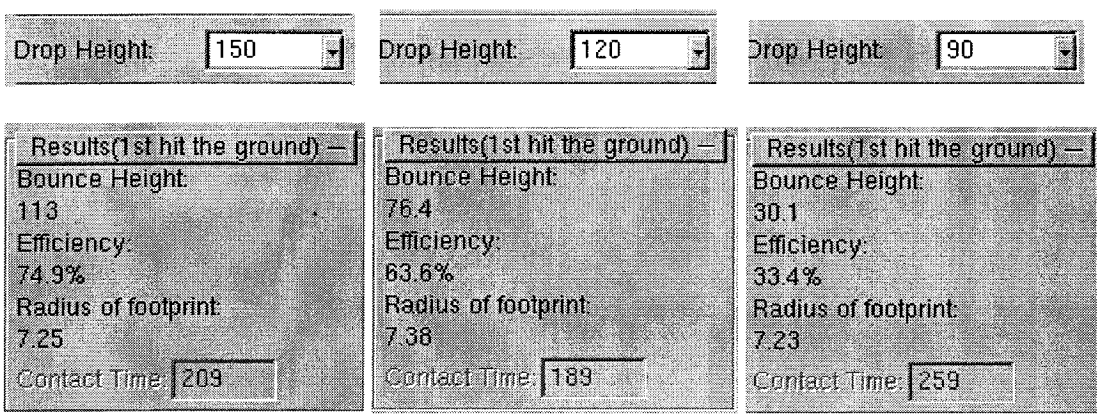


FIGURE 5-9 RESULTS WITH DIFFERENT DROP HEIGHT

6. Conclusion

After discussing all the research, the technical aspects and development of this project, we come to conclusions and some suggestions for future work of the project.

6.1 Experiences in Physics-based Approaches

Having reviewed our work, we can summarize some achievements and shortcomings in applying physics-based approaches in this project:

Achievements:

We achieved in physics-based modeling, and physical-based calculation using physical processes for realistic simulation of the motion of a deformable squashball, which will be difficult to simulate with purely geometric techniques. We applied the mass-spring system, a simple physical technique with well-understood dynamics for modeling, and animated real-time with interaction. The results, the processes of the deformation are very close to the shape changes of a real ball. It basically achieved our goal. According to the results, we can say that physical-based methods are very effective for 3D graphic computing. It gives a good approximation of a complicated physical process.

The program is well encapsulated and concise because it hides the 3D details using vector, point and related functions from CUGL.

Limitations:

- In physics-based modeling, numbers present properties of objects in the real physical word. We have seen many variables in chapter 3 such as the constant of pressure K_p. Even if these numbers come from the physical experiments, we need to test and adjust them with the equations of Newton's laws for computer simulations. Take mass-spring system for example: mass-spring model is discrete. It is an approximation of the true physics that occurs in continuous object. The proper values of the spring constants are not always easy to derive from measured material properties.

- One limitation of the physical calculation on the squash ball mesh in this project is: all the force, velocity and displacement computations start from the poles of the ball. In other words, the poles always impact the racquet, the wall and the floor first. We can always rotate the ball before impact, and it will not influence the animation result due to the symmetrical feature of the ball.

6.2 Future Work

In this last section, we describe some suggestions for future work.

Performance

We need to do research to locate the inefficiencies of the program in order to speed up the motion of the ball. In addition, we could test the ball impacting the objects from different directions.

Physical coefficients

Since physical coefficients are experimental values, we need to do more testing to get better values.

Racquet

The racquet is this project is solid, so it is would be worthwhile to implement an elastic racquet. It would be a considerably more complex to simulate a deformable ball impacting an elastic racquet (spring lattice) for the reason that it is difficult to find the contacting vertices.

Game Simulation

Beyond the scope of this project, it would be exciting and challenging to develop a game-like user interface with players and score board.

References

OpenGL

[1] Peter Grogono, Lecture Notes of COMP 6761 Advanced Computer Graphics, Concordia University, 2002

[2] Peter, Grogono, Getting Started with OpenGL, Concordia University, 2002

[3] F.S.Hill, JR., Computer Graphics Using OpenGL, Prentice Hall, second edition, 1990, p292

Physics

[4] Sarah F.F. Gibson and Brain Mirtich, A Survey of Deformable Modeling In Computer Graphics, Massachusetts, 1997

[5] Paul Bourke, Particle System Example, February 1998, http://astronomy.swin.edu.au/~pbourke/modelling/particle

[6] Tom Henderson, The Physics Classroom, 1996-2004, http://www.glenbrook.k12.il.us/gbssci/phys/Class/newtlaws/u2l1a.html

Squash

[7] Racquet Research, 2002, http://www.racquetresearch.com

[8] Slow-motion playback of racquetball shot and bounce,

http://www.engr.colostate.edu/~dga/high_speed_video/#Sports_Equipment

[9] World Squash Federation, http://www.squash.org/

[10] Squash Player, http://www.squashplayer.co.uk/

Source

[11] Concordia University Graphics Library, Concordia University, May 2004, http://www.cs.concordia.ca/~grogono/CUGL

[12] GLUI User Interface Library, version 2.2, November 2004, http://www.nigels.com/glt/glui/

Related Work

[13] M. James Bridge, "The way balls really bounce", Physics Education, Issue 4, p.236-241, July 1998

[14] "The way balls bounce," 2D simulation of the bounce of an air-filled ball, http://www.xmas.demon.co.uk/physics.html

[15] Maciej Matyka, "How to implement a pressure soft body model", March 2004, http://panoramix.ift.uni.wroc.pl/~maq/soft2d/index.php,

[16] Maciej Matyka and Mark Ollila, "Pressure model of softball simulation", 2003

[17] Jessica Fitch and J. Reed Walker, "Deformable ball simulation", April 2005, http://vorlon.cwru.edu/~jrw24/CG/Report.htm

[18] Demetri Terzopoulos, John Platt, Alan Barr and Kurt Fleischer, "Elastically deformable models", Computer Graphics, volume 21, number4, July 1987

[19] Demetri Terzopoulos and Dimitri Metaxas, "Dynamic deformation of solid primitives with constraints", ACM SIGGRAPH Computer Graphics, volume 26, issue 2, July 1992

[20] Kazuya G. Kobayashi and Katsutoshi Ootsubo, "t -FFD: free-form deformation by using triangular mesh", Proceedings of the eighth ACM symposium on Solid modeling and applications, 2003