Rag doll physics simulation

Victor French

Friday, August 17, 2012

Abstract

Natural movement of a characters body is currently one of the most important requirements in computer games and a large amount of animation for films and television. However, the most important implementation is in computer games. If realistic movement is not done correctly it may take the interest away from players and games, because it is what catches the interest of users and it maintains that attentiveness during the whole storyline.

Its basic application is done by using ragdoll dynamics that will help in simulating the realistic fall and reaction that a body might have while reacting to its environment and forces that are being applied to it. Its implementation consists of using rigid body dynamics to represent the bones and joints of the ragdoll that will allow the game to realistically simulate the collision of the body with the scene and events occurring inside the game.

There has been a lot of work done in ragdoll simulations in the past which has improved with time and has allowed the quality of games that we can see today. The correct collision detection, impulses on collision and joint constraints on movement are the most important characteristics of ragdoll dynamics. This includes the correct use of physics and the three dimensional mathematical calculations it involves.

Acknowledgment

I would like to start by recognizing the great education I have received in this institution and thank the NCCA staff for all the knowledge I have received during this year of studies which will help me grow in the industry. But more specifically I want to express gratitude to the following people:

- Professor Jon Macey, who gave me a lot of insight into the world of programming and scripting in CG.
- Mathieu Sanchez, who has been there for all of us during this year to help us with any problems and concerns during our assignments.
- Ben Kenwright, whom implemented his own ragdoll dynamics and was kind enough to share his research and kept in contact to make sure that I was progressing without problems with my project, and offer help to solve them.

Contents

1. Introduction **9**

2. Related Work **10**

3. Newton's laws of motion **11**

3.1. The first law of motion 11

3.2. The second law of motion 11

3.3. The third law of motion 13

4. Laws of force **13**

4.1. Gravitational force 14

4.2. Force of friction 14

4.2.1. Static friction 15

4.2.2. Kinetic friction 15

4.2.3. Coefficients of friction 15

4.3. Spring force 16

4.3.1. Hooke's law 16

4.3.2. Limit of elasticity 17

4.4. Collision force 17

4.4.1. Closing and separating velocities 18

4.4.2. Coefficient of restitution 19

4.4.3. Collision with immovable bodies 19

4.4.4. Impulses 20

4.4.5. Collision Detection 21

4.4.5.1. Resolving Interpenetration 21

4.5. Types of contact 22

4.5.1. Point contact 22

4.5.2. Edge contact 22

4.5.3. Face contact 23

5. Ragdoll Dynamics 23

5.1. Bone 24

5.2. Joint 24

5.2.1. Ball Joint 24

5.2.2. Hinge Joint 25

5.2.3. Constrained Joint 25

6. Implementation Design 26

6.1. UML Diagram 26

6.2. Class Diagrams 27

6.2.1. Collision Resolution 27

6.2.2. Force Management 30

7. Conclusion........ 31

7.1. Limitations 31

7.2. Future work 31

Bibliography 32

List of Equations

Equation 1: Force applied to a bodies mass 11

Equation 2: Acceleration of a bodies mass 11

Equation 3: Acceleration using length and time 12

Equation 4: Integration to calculate new velocity 12

Equation 5: Integration to calculate new position 12

Equation 6: Gravitational force between two bodies 14

Equation 7: Force for static friction 15

Equation 8: Force for kinetic friction 15

Equation 9: Hooke's law of spring force 16

Equation 10: Change in distance of spring 16

Equation 11: Hooke's law of spring force with change of distance 16

Equation 12: Vector representation of Hooke's law 17

Equation 13: Closing velocity between two bodies 18

Equation 14: Simplified calculation of closing velocity 18

Equation 15: Separating velocity between two bodies 18

Equation 16: Total separating velocity after collision 19

Equation 17: Momentum of collision 19

Equation 18: Separating velocity after collision with immovable body 20

Equation 19: Impulse at collision 20

Equation 20: Change in velocity per individual impulse generated 20

Equation 21: Validate bodies change of position from penetration 21

Equation 22: Relation of distances by mass ratio 22

Equation 23: Distance of change in position after interpenetration 22

Equation 24: Distance of change in position after interpenetration with directional vector 22

List of Figures

Figure 1: Two objects colliding with opposite forces. 13
Figure 2: Two objects traveling at an amount of acceleration from collision. 13
Figure 3: Vector representation of spring force 17
Figure 4: Closing velocity between two bodies 18
Figure 5: Separating velocity between two bodies 19
Figure 6: Separating velocity after colliding with floor 20
Figure 7: Interpenetration between bodies 21
Figure 8: Point contact between two bodies 22
Figure 9: Edge contact between two bodies 23
Figure 10: Face contact between two bodies 23
Figure 11: Degrees of freedom of ball joint 25
Figure 12: Degree of freedom of hinge joint 25
Figure 13: No degrees of freedom from constrained joint 26

List of Tables

Table 1: Static and kinetic coefficients of friction 16

1. Introduction

Computer Graphics have come a long way since the first films and video games developed using digital effects, animation and every other computer generated applications applied in these areas to improve the quality offered to the viewers and users of these tools.

One of the main uses applied to computer graphics is into the area of video games which has had a big success in its implementation. The area of electronic games started with a solely electronic game in 1947, when two individuals named Thomas T. Goldsmith, Jr. and Estle Ray Mann developed the "cathode ray tube amusement device." that was based on recording and controlling the quality of an electronic signal. It was never put into production or sold to the public, so it was not given much attention.

The real attention to games started in the 1970s which was when arcade games, computer games and consoles were announced to the public. It started by being only two dimensional games and in that time has evolved into three dimensional games, as well as a better quality of products.

It has been research, development and the industries rise of expectancy in the quality and realism of its products that has made for an accurate and effective improvement in the area of games. One of the improvements considered important included finding a way to make human body movements more realistic in games without requiring long hours of work analyzing all of the positions expected in realistic game play, which would require animating all of the movements and rendering them in order to have a complete and accurate game to offer the public.

To make this possible the concepts of artificial intelligence and rigid body dynamics were introduced. The first term includes using the machine's tools to automatically analyze the possible results in the game by using intelligent agents that perceives the environment and increases the success of the final result for realism. The term for rigid body dynamics consists on capturing the collision of an object with its environment, implementing accurate forces and impulses to it for a more realistic movement.

This project will concentrate on implementing the later concept to create a simulation of a ragdoll which consists in connecting a group of rigid bodies to represent the different bones in the body an simulate the collision of a basic representation of a body by using third dimension mathematics and physics to simulate an accurate interaction. The necessary mathematics and important physics required will be explained and explained to give a better understanding of how it works and how to effectively achieve this.

2. Related Work

Ragdoll dynamics was originally introduced as a way to procedurally simulate physically based animations of bodies falling to represent the event of a characters death in an animated film or video game. Without the stiffness of the muscles of the human body it leads the body to react less like a real human body and more like a doll, leading into comical, unlikely or compromising positions.

There are cases that have concentrated in only one simple implementation of a ragdoll using the very basic requirements of a ragdoll implementation to test the theory of ragdoll physics (Hennix, et al., 2003) by building a simple simulation. More in depth work has been published by individuals like Ben Kenwright (2012) whom took a more complicated approach into joint constraints and implemented more realistic joint forces to constraint the movement of the bones on a skeletal structure for a ragdoll made out of rigid bodies.

Related to this topic, there have been various implementations developed to create analyze and attempt to improve the implementation of ragdoll simulations by studying the requirements and looking at possible improvements like the case of Stefan Glimberg et al. (2007) who wrote a paper where they explain the importance of properly testing the functionality of rigid bodies and joint constraints in a ragdoll.

They go so far as to demonstrate and compare the different ways they tested the constraint of movement for the rigid bodies in order to find the most accurate option with the purpose of making their findings available for researchers to find and take into consideration when implementing their own simulation.

There have also been books written on creating a functional physics engine for games to document the requirements of an accurate physics engine. One example of this is the book written by Ian Millington (2007) in which he accurately explains the basic mathematical functions, laws of physics and 3D mathematical data necessary for a well-made physics engine and goes on to slowly build up from explaining the most basic to very complicated implementations of physics for games.

Similar to this, implementations have been developed following the same logic (Garstenauer, 2006) and understanding the basic theory required for a proper ragdoll implementation and performing various other implementations to test the correct physics and making sure that every new addition to the implementation is efficient and properly validated.

There are cases in which this is integrated into a bigger implementation that includes other more accurate tools like artificial intelligence to determine a more realistic and life like motion of the human body (Arikan, et al., 2005) in which they develop an artificial intelligence to help make decisions depending on the events happening in the scene and make tests to analyze the results obtained from the research.

3. Newton's laws of motion

Issac Newton founded and proved three important rules in order to correctly explain the motion of an object in a given space. These rules are the pillar of a correct simulation of a moving body interacting with the environment, because they explain how the mass of the body and the forces applied to it will be combined to give an accurate movement of the body in a given direction depending on where the force was applied. This includes a change of position of the body and its rotation by force.

3.1. The first law of motion

To start with, *mass* is the amount of resistance that a body executes in order to prevent from being accelerated. In physics, the resistance performed by the body is referred to as *inertia* and the magnitude needed to affect it and produce acceleration on the body is known as *force*. The first law of motion is represented by the resistance of the body to forces:

> *"Every body persists in its state of being at rest or of moving uniformly straight forward, except insofar as it is compelled to change its state by force impressed." (Dunn & Parberry, 2011)*

3.2. The second law of motion

The second law of motion explains that the net force applied to a body is relative to the amount of acceleration applied to the mass of the body which causes the body to move in the given direction. This is shown in ***Equation 1*** (Dunn & Parberry, 2011).

$$\boxed{\vec{F} = m\vec{a}}$$

EQUATION 1: FORCE APPLIED TO A BODIES MASS

Where $\vec{\boldsymbol{F}}$ represents the force $\boldsymbol{m}$ is the mass of the body and $\vec{\boldsymbol{a}}$ is the acceleration applied. As a real world example we can take the acceleration to be 9.80665 m/s^2 (the acceleration due to gravity) and we can consider the mass of the body to be 2.5 kg the solution to this would be:

$$\vec{F} = 2.5 \times 9.80665$$
$$\vec{F} = 24.516625$$

This means that the force of gravity necessary for the body to travel in the direction of the center of the Earth is 24.516625 Newtons (N). The same thing applies for the calculation of acceleration given the magnitude of force and quantity of mass of a body which is represented by ***Equation 2*** (Dunn & Parberry, 2011).

$$\boxed{\vec{a} = \frac{\vec{F}}{m}}$$

EQUATION 2: ACCELERATION OF A BODIES MASS

Which means that by using the magnitude of force previously obtained and the mass given we can prove that the acceleration given is true to the law of motion.

$$\vec{a} = {}^{24.516625}/_{2.5}$$
$$\vec{F} = 9.80665$$

In order for these concepts to be implemented in 3D space we need to analyze the requirements to represent these values in our application. Considering that mass is a numerical value representing the amount of resistance to acceleration its representation will be a scalar quantity. Both acceleration and force require a scalar quantity as the magnitude that will tell the application how much will be applied, but they also require a direction towards which the magnitude will be applied. So its implementation will be implemented by a vector that will hold the x, y and z axis for the direction and the magnitude that will be applied towards each dimension.
However, in order to correctly implement this equation a fourth value needs to be considered. Even though in the real world this happens during continuous time, in order to be accurately simulated a time frame need to be considered for the simulation and to be able to visualize the steps in the simulation so the equation needs to be divided further.
By definition of acceleration is the magnitude of length ($\boldsymbol{L}$) in direction over a given time frame ($\boldsymbol{t}$) squared which is symbolized by ***Equation 3*** (Dunn & Parberry, 2011).

$$\boxed{\vec{a} = \frac{L}{t^2}}$$

Equation 3: Acceleration using length and time

This means that by using integration in ***Equation 3*** to get the new velocity ($\overrightarrow{\boldsymbol{v}'}$) of that body's mass over time ($\boldsymbol{t}$) we need to add the acceleration ($\vec{\boldsymbol{a}}$) current velocity ($\vec{\boldsymbol{v}}$), visualized in ***Equation 4*** (Millington, 2007).

$$\boxed{\overrightarrow{\boldsymbol{v}'} = \vec{\boldsymbol{v}} + \vec{\boldsymbol{a}}\boldsymbol{t}}$$

Equation 4: Integration to calculate new velocity

The equations defined in this section help in understanding how these calculations will affect the position of the body given the calculation for acceleration from the applied force seen in ***Equation 2*** by using integration for position. This brings us to ***Equation 5*** (Millington, 2007), needed for the motion of a given body that results in the new position ($\overrightarrow{\boldsymbol{p}'}$) by adding the obtained velocity ($\vec{\boldsymbol{v}}$) over time ($\boldsymbol{t}$) to the current position ($\vec{\boldsymbol{p}}$).

$$\boxed{\overrightarrow{\boldsymbol{p}'} = \vec{\boldsymbol{p}} + \vec{\boldsymbol{v}}\boldsymbol{t}}$$

Equation 5: Integration to calculate new position

3.3. The third law of motion

The last law of motion is directed at the collision between two bodies which causes them to receive resulting forces on each other at opposite directions. The forces applied are of the same magnitude towards both directions, but the magnitude of change in position depends on the mass of both bodies. This means that the body with the smallest mass will receive a greater velocity, while the biggest mass will receive a smaller amount of *velocity*. This is represented in ***Figure 1*** *and* ***Figure 2***.

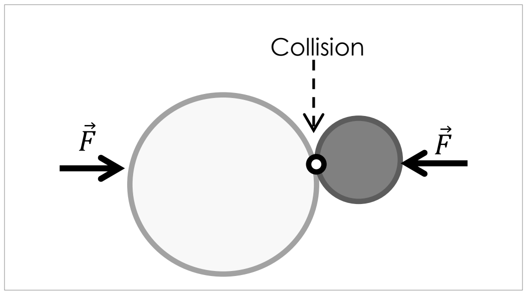


FIGURE 1: TWO OBJECTS COLLIDING WITH OPPOSITE FORCES.

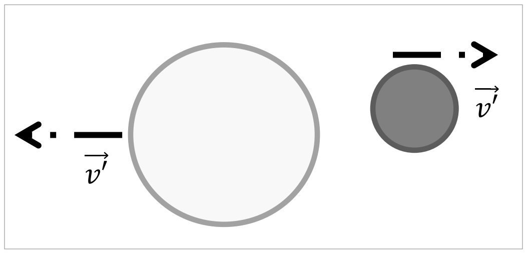


FIGURE 2: TWO OBJECTS TRAVELING AT AN AMOUNT OF ACCELERATION FROM COLLISION.

Because the forces applied in the moment of impact are not constant forces they can be considered an *impulse* on collision at a given contact, like seen in ***Figure 1***. Since the *impulse* is performed only once at a given point in time there is actually no *acceleration* of *mass* so the impact performs a direct change in *velocity* instead.

4. Laws of force

We previously mentioned forces acting on a body to accelerate it and create motion and there are many different forces that could be applied to an object at any given time. Some of these forces are constant like gravity and wind, and some of them are dependent on an event like friction, spring forces, buoyancy and collisions with other bodies.
The most common forces present in a ragdoll simulation are gravity, friction, spring and collisions forces, so this section will concentrate on giving a better understanding of how they work and what is needed for a correct calculation in order to be implemented in a physically accurate simulation.

4.1. Gravitational force

Even though Newton is better known for his three laws of motion, he also identified several other laws of physics. One of which is the *law of universal gravitation*, which describes gravitational force to be a force of attraction between bodies that causes a body to receive a force that attempts to push them to the center of mass of the other body. As stated by Dunn et al. (2011) the force applied for gravity is "proportionate to the product of their masses and inversely proportionate to the square of the distance between the objects", which can be calculated by using ***Equation 6*** (Dunn & Parberry, 2011).

$$\boxed{\vec{F} = G\frac{m_1 m_2}{r^2}, \qquad G = 6.67 \times 10^{-11}}$$

EQUATION 6: GRAVITATIONAL FORCE BETWEEN TWO BODIES

Where $\boldsymbol{\vec{F}}$ is the magnitude of force, $\boldsymbol{m_1}$ is the mass of the first body, $\boldsymbol{m_2}$is the mass of the second body, $\boldsymbol{r}$ is the distance between their centers of mass and $\boldsymbol{G}$ is the physical constant of the universe which is specified above.
Because this is an equation for universal purposes and is only useful for calculations that require great quantities of mass like planetary motion or ocean tides like explained by Dunn et al. (2011), we are going to take Earth's surface as the main constant value for the first mass for its body.
This means that since we are going to concentrate on earth's surface interacting with objects and most of the objects are within a certain height from the surface, it is to be assumed that the distance between both centers of mass will not be changing drastically. So the distance that is commonly used for physical calculations is also constant.
Since we have most values necessary for the gravitational force we can imply that in this case the value for the necessary acceleration for the mass to reach the expected gravitational force is the known value for reaching the necessary value for acceleration due to gravity (9.80665 m/s^2), see ***Equation 1***.

4.2. Force of friction

If we consider a body resting on a surface and another body or some external force suddenly performs the resting bodies mass to accelerate in a perpendicular direction to the surface it will slide across the surface without being able to stop considering that once a body is in acceleration it requires an opposite force to neutralize the given acceleration to cause it to discontinue.
Supposing that there are no other objects resting on the surface the body won't be able to stop unless there was an external force that would perform the deceleration of a body in motion on a given surface. Well, the force required to do that is called ***kinetic friction***, which is one of two forces of friction necessary for a body resting on a surface. It is defined as the counteracting force to a force being applied to a body in movement on a surface (Dunn & Parberry, 2011).
The second existing frictional force is the ***static friction***, which according to Dunn et al. (2011) is the force that reaches a maximum amount to counteract any force being applied to a resting

body on a surface, after which the body will start to move across the surface by the amount of force resulting from the difference in forces applied to the mass of the body.

4.2.1. Static friction

In order to calculate the amount of force for static friction we required two values: the coefficient of static friction ($\boldsymbol{\mu_s}$) and the magnitude of the normal force ($\boldsymbol{n}$) in the direction perpendicular to the surface the body is resting on, which we can visualize in ***Equation 7*** (Dunn & Parberry, 2011).

$$\boxed{\overrightarrow{f_s} = \mu_s n}$$

EQUATION 7: FORCE FOR STATIC FRICTION

4.2.2. Kinetic friction

As with static friction, kinetic friction is calculated by taking the normal force ($\boldsymbol{n}$), but the coefficient of static friction would no longer apply to calculate the force necessary for kinetic friction. So we would need a new value called the coefficient of kinetic friction ($\boldsymbol{\mu_k}$) applied to ***Equation 8*** (Dunn & Parberry, 2011).

$$\boxed{\overrightarrow{f_k} = \mu_s n}$$

EQUATION 8: FORCE FOR KINETIC FRICTION

4.2.3. Coefficients of friction

Both ***Equation 7*** and ***Equation 8*** use the same normal force to apply the necessary friction to counteract the forces being applied to the same body, so it is only logical to assume that the difference between both types of friction are their coefficients. The coefficients for a specific body differ depending on the material from which both the body and the surface it is resting on are made of. ***Table 1*** (Dunn & Parberry, 2011) has been included bellow to give an example of possible values for both coefficients of friction depending on the combination of materials between the body and the surface.

Material 1	**Material 2**	$\boldsymbol{\mu_s}$ **(Static)**	$\boldsymbol{\mu_k}$ **(Kinetic)**
Aluminum	Steel	0.61	0.47
Copper	Steel	0.53	0.36
Leather	Metal	0.4	0.2
Rubber	Asphalt (dry)	0.9	0.5 – 0.8
Rubber	Asphalt (wet)		0.25 – 0.75
Rubber	Concrete (dry)	1.0	0.6 – 0.85
Rubber	Concrete (wet)	0.30	0.45 – 0.75
Steel	Steel	0.80	
Steel	Teflon	0.04	
Teflon	Teflon	0.04	
Wood	Concrete	0.62	

Wood	Clean metal	0.2 – 0.6	
Wood	Ice	0.05	
Wood	Wood	0.25 – 0.5	
Wood (waxed)	Dry snow		0.04

TABLE 1: STATIC AND KINETIC COEFFICIENTS OF FRICTION

4.3. Spring force

Spring forces have a lot of uses in obtaining accurate and impressive effects for objects. As described by Ian Millington (2007) "Springs and particles alone can produce a whole range of impressive effects such as ropes, flags, cloth garments, and water ripples." A particle in a 3D world can be represented by a single point in a specified position. A spring in the other hand is a representation of two or more connected particles that are limited by a specified maximum and/or minimum distance from which the magnitude of the particles distances may not exceed. The calculation for the force of a spring is described by **Hooke's Law.**

4.3.1. Hooke's law

According to Robert Hooke, the force (see ***Equation 9***) exerted by a spring is relies on the distance it is stretched or compressed compared to the position where the spring is considered to be resting (Millington, 2007).

$$\boxed{\vec{f} = -k\Delta l}$$

EQUATION 9: HOOKE'S LAW OF SPRING FORCE

Where $\Delta \boldsymbol{l}$ is the change in distance at extension or compression of the spring for each time step and $\boldsymbol{k}$ is the spring constant, which defines how much stiffness the spring has in order to prevent it from applying too much force to it.
The change in distance ($\Delta \boldsymbol{l}$) from the applied force is basically the difference from the position of rest which can be represented by $\boldsymbol{l_0}$ and the current position with the symbol $\boldsymbol{l}$. See ***Equation 10.***

$$\boxed{\Delta l = l - l_0}$$

EQUATION 10: CHANGE IN DISTANCE OF SPRING

This leads us to ***Equation 11*** (Millington, 2007), which is the complete formula to get the spring force defined by Hooke's law.

$$\boxed{\vec{f} = -k(l - l_0)}$$

EQUATION 11: HOOKE'S LAW OF SPRING FORCE WITH CHANGE OF DISTANCE

Finally, considering that up to now the calculations for the spring force have been in only one dimension we need to figure out how to get the representation for Hooke's law in more than one dimension. So, in order to do this we need to represent the positions as vectors with the dimensional axis for each coordinate. For that we need to consider ***Equation 12*** (Millington, 2007)**.**

$$\boxed{\vec{f} = -k(|d| - l_0)\hat{d}}$$

Equation 12: Vector representation of Hooke's law

Where d is the vector representation of the difference between the destination point or the other end of the spring, and the point of origin of the spring or the point connected to the body for which we are generating a force. Which leaves us to believe that $|d|$ is the scalar value representing the length between those two points and $\hat{d}$ is the normalized difference to determine the direction in which the force will be applied. (See ***Figure 3***)

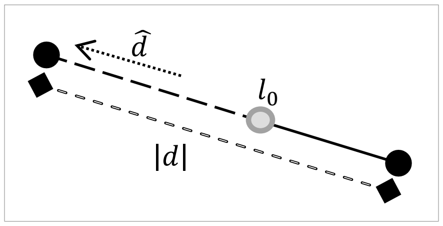


Figure 3: Vector representation of spring force

4.3.2. Limit of elasticity

The limit of elasticity refers to the maximum and minimum amount permitted for the spring to expand or compress before it stops being a spring and loses its functionality. An example specific to ragdoll simulation is the implementation of the joints connecting the bones which we will refer to in a later section.
In this case we could make the joints break at the moment it reaches its maximum limit of elasticity and remove the connection between bones, causing the bone that is no longer connected to the rest of the ragdoll to fly away by the force generated on impact to represent the body losing a limb.

4.4. Collision force

A contact force is the force applied to a body when touching another body while in motion. It is important to note that there are three types of contacts (Millington, 2007) two bodies can have between each other: point, edge and face contact. These contact types will be explained in detail in this section, but for now it is important to understand that the type of contact occurring during a simulation depends on the shape of the body, the angle of each body and the position each body has during the moment of contact.

When two bodies touch during motion, the resulting motion after contact can be calculated from their movement before touching. This calculation of the movement on contact is called collision resolution (Millington, 2007). Even though the contact between both bodies does not happen at large velocities in this case the contact occurring is considered a collision because both bodies are still transmitting forces between each other that causes them to rebound on the direction of contact to a certain magnitude in distance. For a better understanding of how this is computed we will explain in more detail and go through the steps necessary to successfully achieve an effective collision simulation between bodies.

4.4.1. Closing and separating velocities

As defined by Ian Millington (2007), the total speed at which two objects move together is known as closing velocity. This calculation uses a scalar value and direction without needing to use a vector representation. This is done by taking the sign of the scalar value to determine the direction where the velocity will be applied. See ***Equation 13***.

$$\boxed{v_c = \overrightarrow{v_a} \cdot \widehat{(p_b - p_a)} + \overrightarrow{v_b} \cdot \widehat{(\overrightarrow{p_a} - \overrightarrow{p_b})}}$$

EQUATION 13: CLOSING VELOCITY BETWEEN TWO BODIES

Where the closing velocity of the two bodies is $\boldsymbol{v_c}$, $\overrightarrow{p_a}$ is the position of body a and $\overrightarrow{p_b}$ is the position of body b. The values $\overrightarrow{v_a}$ and $\overrightarrow{v_b}$ are the velocities for body a and b respectively. According to ***Equation 13***, in order to obtain the closing velocity of both objects in scalar form we take the velocity of body a and apply a dot product (also known as scalar product of a vector) of the vector of the velocity with the resulting vector of the difference in position between body b and body a from body a's point of view and then proceed to do the same for body b's point of view and add both values together to get the resulting velocity. A more simplified representation would be the one shown in ***Equation 14*** (Millington, 2007).

$$\boxed{v_c = -(\overrightarrow{v_a} - \overrightarrow{v_b}) \cdot \widehat{(\overrightarrow{p_a} - \overrightarrow{p_b})}}$$

EQUATION 14: SIMPLIFIED CALCULATION OF CLOSING VELOCITY

According to ***Equation 14,*** when two bodies close on each other the resulting relative velocity will be negative, and when they are separating away from each other it will be positive. So, by changing the sign from ***Equation 14*** we would get a separating velocity (v_s) represented in ***Equation 15*** (Millington, 2007).

$$\boxed{v_s = (\overrightarrow{v_a} - \overrightarrow{v_b}) \cdot \widehat{(\overrightarrow{p_a} - \overrightarrow{p_b})}}$$

EQUATION 15: SEPARATING VELOCITY BETWEEN TWO BODIES

For a visual representation of both types of relative velocities between two bodies refer to ***Figure 4*** and ***Figure 5***.

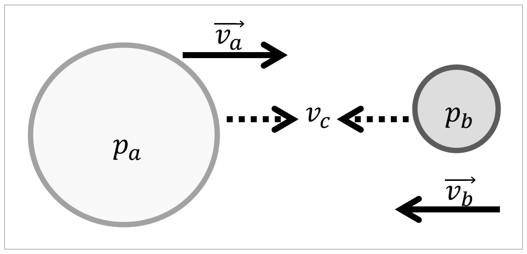


FIGURE 4: CLOSING VELOCITY BETWEEN TWO BODIES

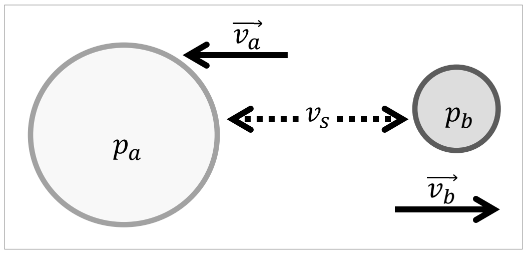


FIGURE 5: SEPARATING VELOCITY BETWEEN TWO BODIES

4.4.2. Coefficient of restitution

As described by Ian Millington (2007), the coefficient of restitution manages the speed in which two bodies separate after collision. So if the value is similar or equal to 1 the separating velocity applied after collision will be the same as the separating velocity before collision, which means that both bodies separate using the entire velocity calculated. In the case where the value for the coefficient is minimal there will be very little separating velocity after collision which means that both objects will attempt to travel together and fuse making it look similar to two objects with a very sticky or force absorbent material. (See ***Equation 16***)

$$\boxed{\overrightarrow{v'_s} = -c\overrightarrow{v_s}}$$

EQUATION 16: TOTAL SEPARATING VELOCITY AFTER COLLISION

Where $\overrightarrow{v'_s}$ characterizes the separating velocity after collision, the separating velocity before collision is represented by $\overrightarrow{v_s}$, and c is the coefficient of restitution.

In order to prove that the obtained separating velocity after collision is correct we have to take into account that the bodies always have to maintain momentum. So, to prove that we need to consider that momentum is when the sum of the mass of each body times their velocities before colliding with each other has to be the same as the sum of the mass of each body times their velocities after collision, as seen in ***Equation 17*** (Millington, 2007).

$$\boxed{m_a\overrightarrow{v_a} + m_b\overrightarrow{v_b} = m_a\overrightarrow{v'_a} + m_b\overrightarrow{v'_b}}$$

EQUATION 17: MOMENTUM OF COLLISION

4.4.3. Collision with immovable bodies

Since a single body is not always going to collide with bodies that are going to be difficult to physically simulate because of their large masses which will make them immovable against very small masses in comparison, there has to be a calculation capable of obtaining the velocity of separation with only one body being affected.

This means that the previously defined calculation for the direction of the separating velocity of two bodies ($\widehat{p_a - p_b}$) will change, because we are no longer able to take the position of a second affected body. So this direction which is normally known as the ***collision normal*** or ***contact normal*** has to be given explicitly for a single body colliding with an immovable body and the magnitude of the direction must always be a scalar value of 1 (Millington, 2007).

For example, when a single body is colliding with the ground we always know that the body will be falling into the ground and considering that the collision normal must always be given from the first body's point of view, which means that the direction will be in the up vector or y axis (see ***Figure 6***) assuming that the ground is not in a slope as follows:

$$\hat{n} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Considering the values available for this case we can deduce that the calculation for the separating velocity is as seen in ***Equation 18*** (Millington, 2007).

$$\boxed{\overrightarrow{v'_s} = (\overrightarrow{v_a} - \overrightarrow{v_b}) \cdot \hat{n}}$$

EQUATION 18: SEPARATING VELOCITY AFTER COLLISION WITH IMMOVABLE BODY

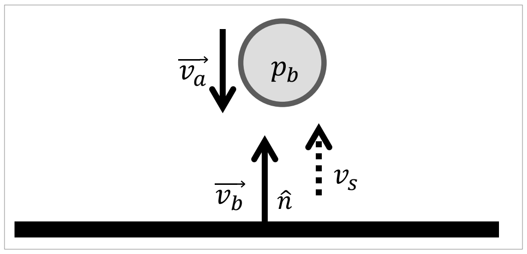


FIGURE 6: SEPARATING VELOCITY AFTER COLLIDING WITH FLOOR

4.4.4. Impulses

When talking about Newton's third law of motion impulse was defined as a force that is only applied once because of it being caused by a sudden change in direction between bodies. Well, that is exactly what an impulse is, an instantaneous change in velocity caused by colliding with another body or an opposing force. In the same way that we deduced the calculation for force in ***Equation 1*** we can do the same for impulse in ***Equation 19*** (Millington, 2007).

$$\boxed{\vec{\imath} = m\vec{v}}$$

EQUATION 19: IMPULSE AT COLLISION

Where $\vec{\imath}$ is the resulting impulse, m is the mass of the body to which the impulse is applied and $\vec{v}$ is the velocity the mass is travelling after collision.
By using D' Alembert's principle, we can calculate the new velocity a body has to take in order to comply with the impulses applied to a body during collisions. Like for example a box can collide with another box after falling on top of it and hit more than one point on the box. Each one of the contacts the boxes have during the moment of collision will generate a separate impulse to force the separation between them. In order to do this we need to look at ***Equation 20*** (Millington, 2007).

$$\boxed{\overrightarrow{v'} = \vec{v}\frac{1}{m}\vec{\imath}}$$

EQUATION 20: CHANGE IN VELOCITY PER INDIVIDUAL IMPULSE GENERATED

4.4.5. Collision Detection

Collision Detection for bodies touching is centered on gathering the information for the bodies' shape, size, position and motion to determine the moment of impact by comparing the data and defining the point of contact between both bodies.
This is done to be able to resolve the interpenetration between bodies and prevent the simulation from looking like the bodies are going through each other and making it look unrealistic.

4.4.5.1. Resolving Interpenetration

When calculating the interpenetration between object we can define if the two objects are just touching or penetrating when the depth of penetration is equal to zero for when they are touching and positive for when they are penetrating by a certain amount. (See ***Figure 7***)

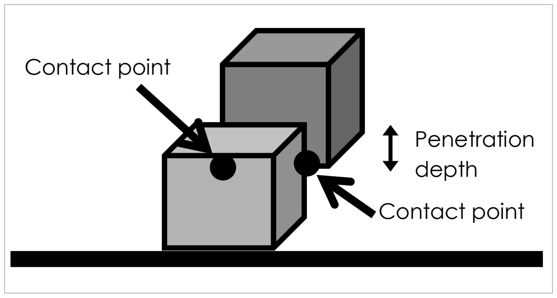


FIGURE 7: INTERPENETRATION BETWEEN BODIES

This means that there will only be need for resolving interpenetration when the depth of penetration is greater than zero, because if they are just touching there is no interpenetration. In order to resolve the interpenetration there has to be an understanding of what is involved during the penetration.
Each body is colliding at a certain force, so in order to resolve the interpenetration and calculate de separating velocity each body has to take it is important to understand that each body has to move apart at inverse proportion to their mass as stated by Ian Millington (2007). This means that a very small mass from a body will be moving at a greater velocity than a ver large body which will slightly change position, depending on the case.
In order to validate the correct movement of both bodies it is important to note that the total change of position by both bodies should be equal to their depth of penetration, as shown in ***Equation 21*** (Millington, 2007).

$$\boxed{\Delta p_a + \Delta p_b = d_p}$$

EQUATION 21: VALIDATE BODIES CHANGE OF POSITION FROM PENETRATION

Where d_p is the depth of penetration, Δp_a is the scalar distance from body a's change in position and Δp_b is body b's distance from its last position. Take into consideration that the ratio of the mass for both bodies indicates that both distances are related as stated by Ian Millington (2007). (See ***Equation 22***)

$$\boxed{m_a \, \Delta p_a = m_b \, \Delta p_b}$$

EQUATION 22: RELATION OF DISTANCES BY MASS RATIO

This leaves us to believe that by combining ***Equation 21*** and ***Equation 22*** we can get the distance for body a as indicated by ***Equation 23*** and ***Equation 24*** (Millington, 2007).

$$\boxed{\boldsymbol{\Delta p_a = \frac{m_b}{m_a + m_b} d_p}}$$

EQUATION 23: DISTANCE OF CHANGE IN POSITION AFTER INTERPENETRATION

To get the direction where body a will travel to, the contact normal ($\hat{n}$) has to be included into the equation. (See ***Equation 24***)

$$\boxed{\Delta p_a = \frac{m_b}{m_{\boldsymbol{a}} + m_b} d_p \hat{n}}$$

EQUATION 24: DISTANCE OF CHANGE IN POSITION AFTER INTERPENETRATION WITH DIRECTIONAL VECTOR

4.5. Types of contact

As mentioned earlier there are 3 types of contacts for bodies in a 3D world: point contact, edge contact and face contact.

4.5.1. Point contact

In a computer generated simulation a point contact is considered the contact that occurs when a body that has a shape with small corners with which it comes into contact with another body, like in ***Figure 8*** (Millington, 2007).

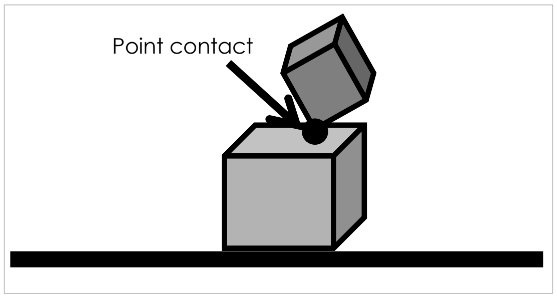


FIGURE 8: POINT CONTACT BETWEEN TWO BODIES

4.5.2. Edge contact

The edge contact refers to a body that touches another contact with two points from its body that represent a side of the colliding body (Millington, 2007). (See ***Figure 9***)

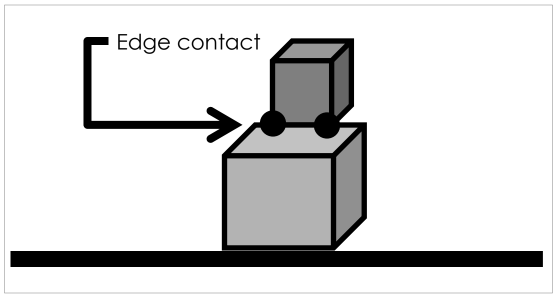


FIGURE 9: EDGE CONTACT BETWEEN TWO BODIES

4.5.3. Face contact

For contacts that occur with bodies that have curved shapes we usually have face contacts because that means that their faces are curved as well. An example of this is a body that has a sphere shape like a tennis ball. At the moment when the ball hits the surface of a table, even though it might look like a point contact because there is only one contact occurring between both bodies, it is known as a face contact because the whole face of the ball is hitting the table, as seen in ***Figure 10*** (Millington, 2007)**.**

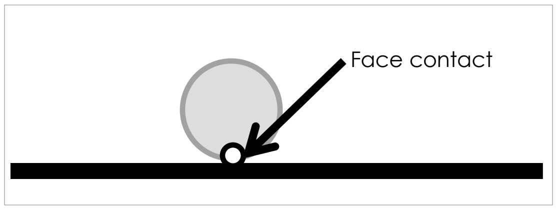


FIGURE 10: FACE CONTACT BETWEEN TWO BODIES

However, curved shaped bodies are not the only ones that can have a face contact with other bodies. Any body may have a face contact depending on the circumstances, but all points of the face have to be in contact with the opposing body at the same time for it to be considered a face contact. As another example, imagine a box hitting the floor flat on one side. This would be considered a face contact, because the four points that a side of a box consists of are in contact with the floor at the same time.

5. Ragdoll Dynamics

Ragdoll Dynamics is the implementation of rigid body dynamics and spring forces that act together to visualize a simulated representation of the skeleton of a human body without the muscular characteristics interacting with its environment visualizing the reaction of the body interacting with external forces present in the simulated environment.
Considering that the body is only constructed with bones and joints that connect the bones together and determine the angular limit of the bones it behaves more like a doll made out of rags and not a real human body, so this type of simulation is known as ragdolls for their similarities.

This is used mainly to create realistic movements of a body that is being thrown around the environment procedurally and eliminating the need of animation for a lot of the actions needed in films or video games.
In this section we will look at the required bone and joint characteristics in order to make an effective ragdoll simulation.

5.1. Bone

Ragdoll dynamics use rigid bodies to represent the bones from which the skeletal structure is made of. These bones can be represented by a number of shapes used for rigid bodies, but the most basic and generally used form is the cube because of its accuracy in detecting collisions. Another reason why it is used so much is because it can fit the desired size needed for the body parts and its faster calculations while colliding with other bodies.
However, this is only true for initial testing and first representations of a rigid body. Most of the people that go on to work more into the topic evolve the application to include more realistic shapes like in the case of Ben Kenwright (2012) and Helmut Garstenauer (2006).

5.2. Joint

Joints are a big part of ragdoll simulations because they act as constraints for the movement of the connected bones of the skeletal structure. Their main function is to limit the amount of rotation the rigid bodies will perform in motion. To correctly implement joints they have to be labeled depending on the freedom of movement allowed so they can be implemented for each bone group depending on the limitations needed. These types of joints in order of freedom are: ball joint, hinge joint and constrained joint as explained by Stefan Glimberg et al. (2007) and Ian Millington (2007).

5.2.1. Ball Joint

A ball joint is characterized for being the joint that has all three rotational degrees of freedom as stated by Stefan Glimberg et al. (2007) which makes it more difficult to restrict since it has to be validated for each rotational degree. It is important to note that in order to restrict the angular motion that the bones will have the angular limit parameter has to be defined at the moment of creation of the constraint.
An example of ball joints in a ragdoll body are: the neck, the shoulders and the hip; which are the parts of the human body that have the most degree of freedom. ***Figure 11*** illustrates how the ball joint connection between bones should function.

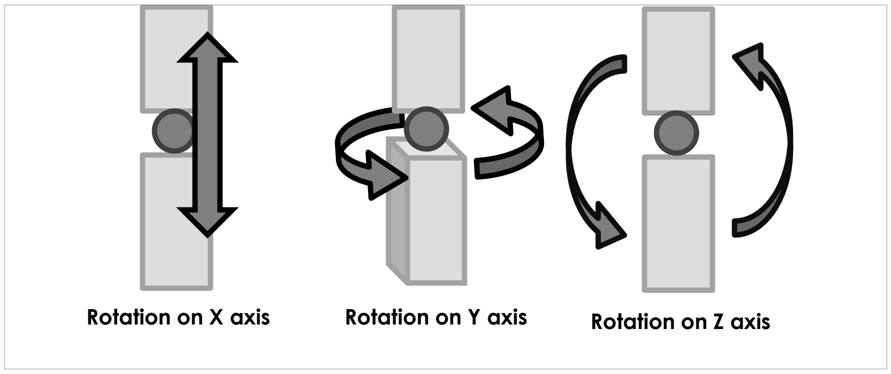


FIGURE 11: DEGREES OF FREEDOM OF BALL JOINT

5.2.2. *Hinge Joint*

This is the type of joint that is more common in a human body due to the fact that it has only one degree of freedom. According to Stefan Glimberg et al. (2007) it is very simple due to it only allowing the bone to move in the direction around the axis where the constraint points are positioned. Similar to the ball joint, for it to work correctly, the joint constraint needs to receive the angular limit that will be applied to it.

To illustrate the angular constraint defined for a hinge joint, ***Figure 12*** has been included bellow.

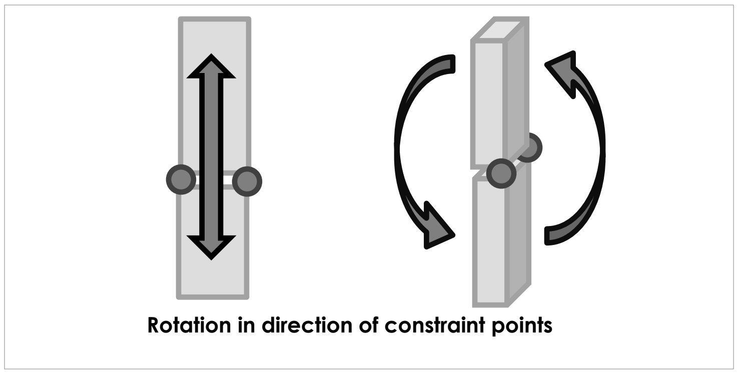


FIGURE 12: DEGREE OF FREEDOM OF HINGE JOINT

5.2.3. *Constrained Joint*

The constrained joint is rarely used because it has no degrees of freedom due to the application of three constraint points that inhibit the object from rotating in any direction. If it were to be used in the simulation of rigid bodies it would be for parts of the body that only need to be connected with the rest of the body and have no other functionality like the ribs, collar bone or any of the smaller bones in the human body, which are normally not needed in this kind of simulation. A visual representation of this would be the illustration seen in ***Figure 13***.

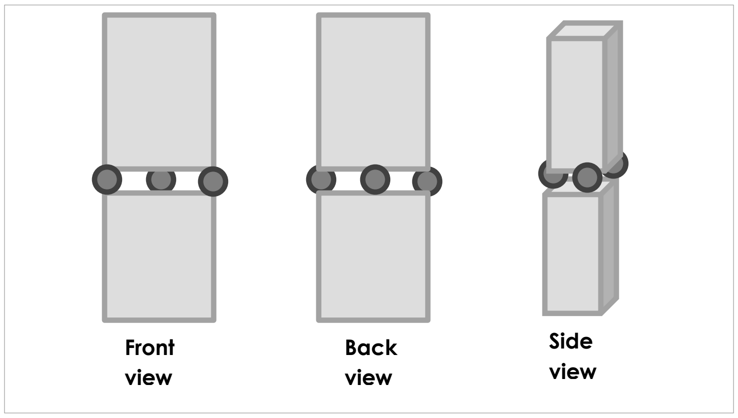
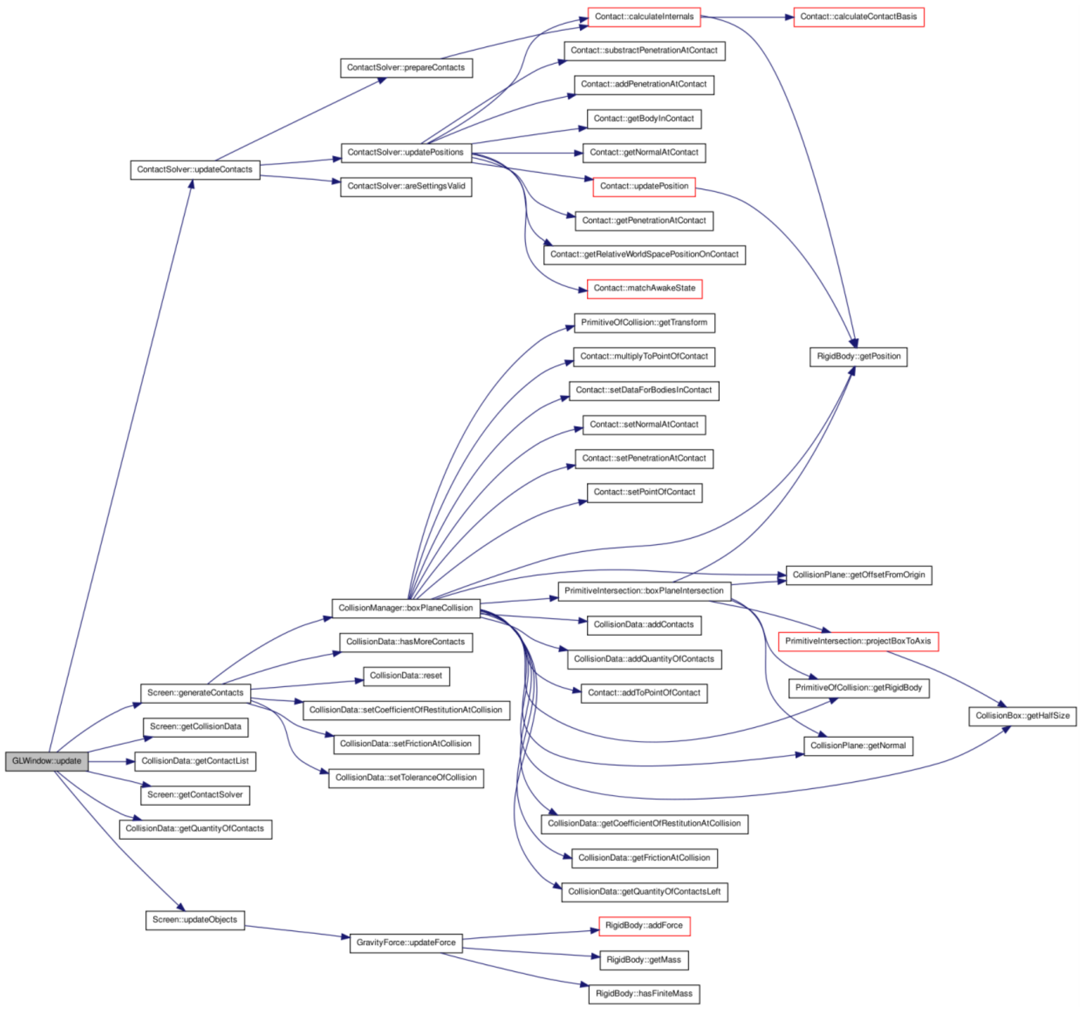


FIGURE 13: NO DEGREES OF FREEDOM FROM CONSTRAINED JOINT

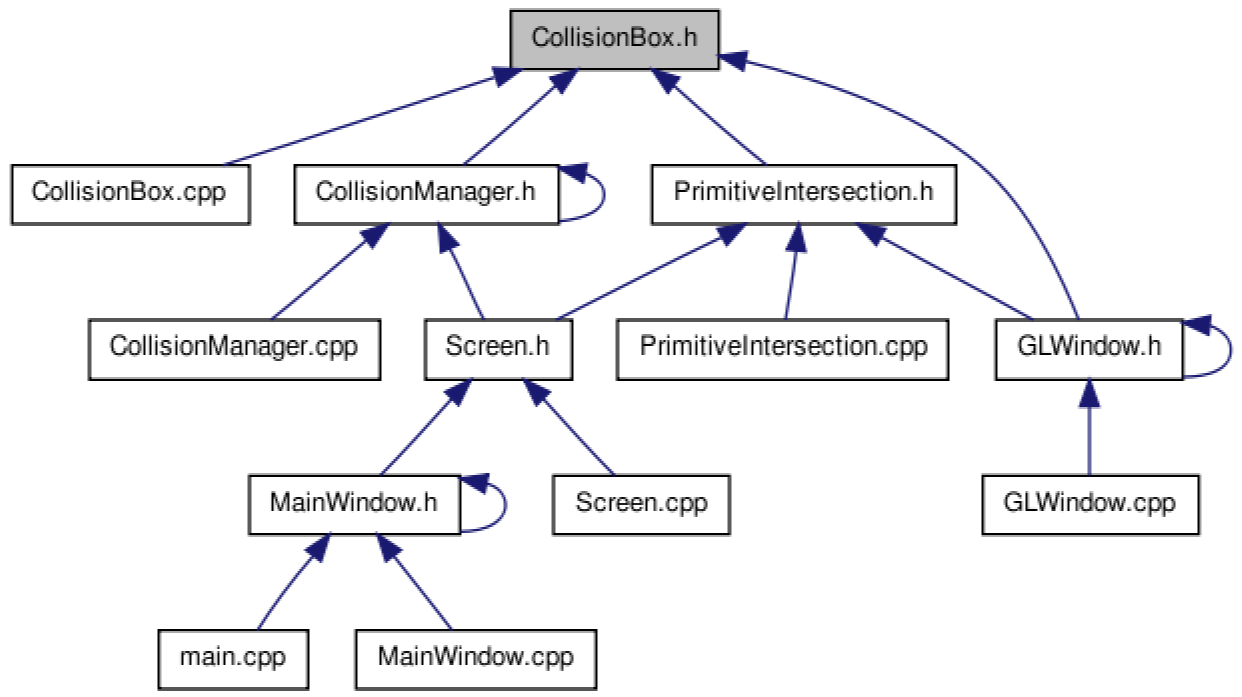
6. Implementation Design

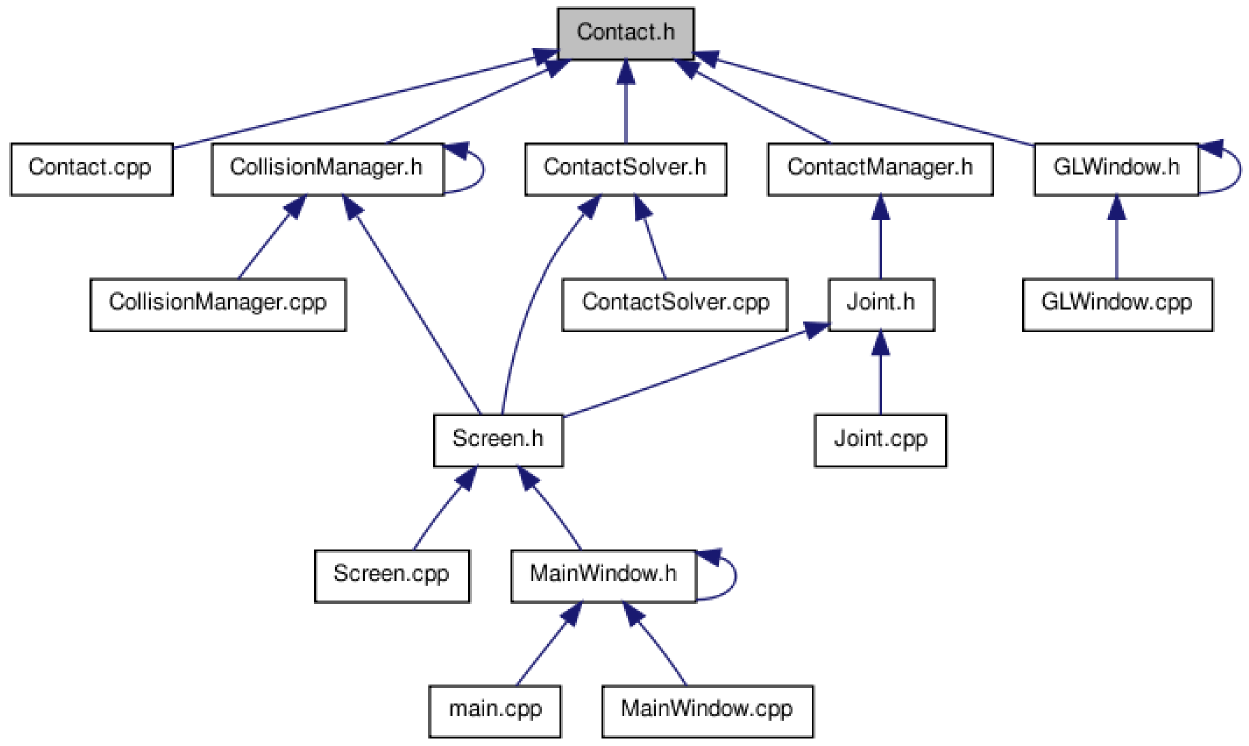
6.1. UML Diagram

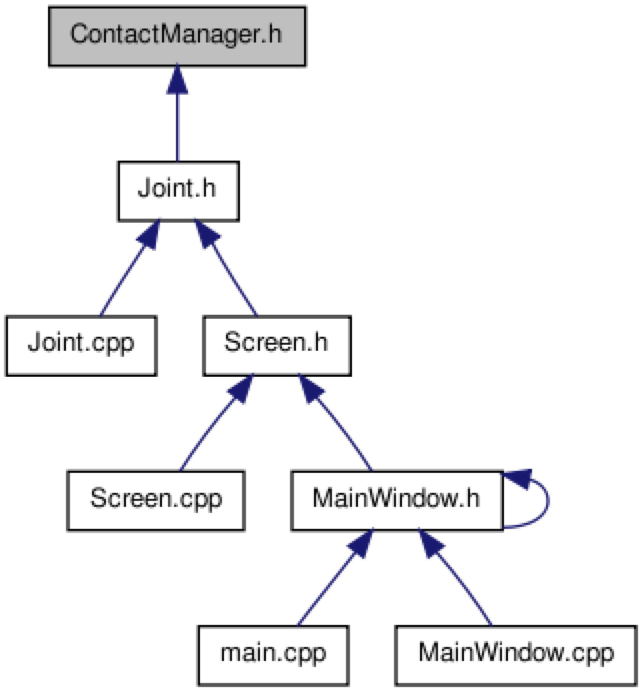

Contact::calculateInternals
Contact::calculateContactBasis
Contact::substractPenetrationAtContact
ContactSolver::prepareContacts
Contact::addPenetrationAtContact
Contact::getBodyInContact
ContactSolver::updatePositions
Contact::getNormalAtContact
ContactSolver::updateContacts
ContactSolver::areSettingsValid
Contact::updatePosition
Contact::getPenetrationAtContact
Contact::getRelativeWorldSpacePositionOnContact
Contact::matchAwakeState
PrimitiveOfCollision::getTransform
Contact::multiplyToPointOfContact
RigidBody::getPosition
Contact::setDataForBodiesInContact
Contact::setNormalAtContact
Contact::setPenetrationAtContact
Contact::setPointOfContact
CollisionPlane::getOffsetFromOrigin
PrimitiveIntersection::boxPlaneIntersection
CollisionManager::boxPlaneCollision
CollisionData::addContacts
PrimitiveIntersection::projectBoxToAxis
CollisionData::hasMoreContacts
CollisionData::addQuantityOfContacts
CollisionData::reset
Screen::generateContacts
Contact::addToPointOfContact
PrimitiveOfCollision::getRigidBody
CollisionData::setCoefficientOfRestitutionAtCollision
CollisionBox::getHalfSize
Screen::getCollisionData
CollisionData::setFrictionAtCollision
CollisionPlane::getNormal
GLWindow::update
CollisionData::getContactList
CollisionData::setToleranceOfCollision
Screen::getContactSolver
CollisionData::getQuantityOfContacts
CollisionData::getCoefficientOfRestitutionAtCollision
CollisionData::getFrictionAtCollision
CollisionData::getQuantityOfContactsLeft
Screen::updateObjects
RigidBody::addForce
GravityForce::updateForce
RigidBody::getMass
RigidBody::hasFiniteMass

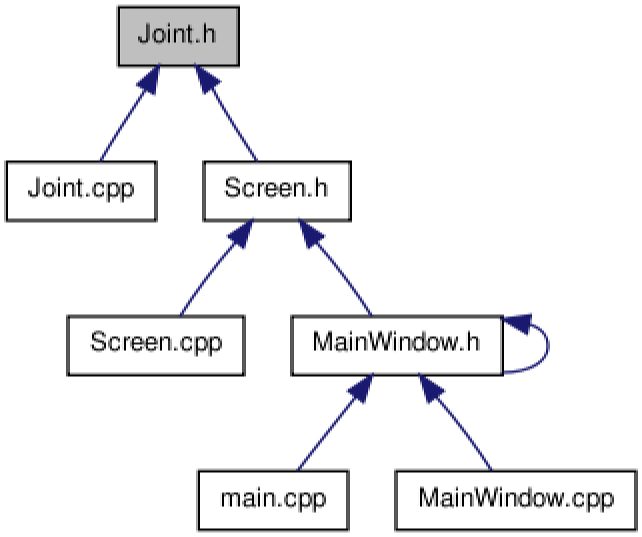
6.2. Class Diagrams

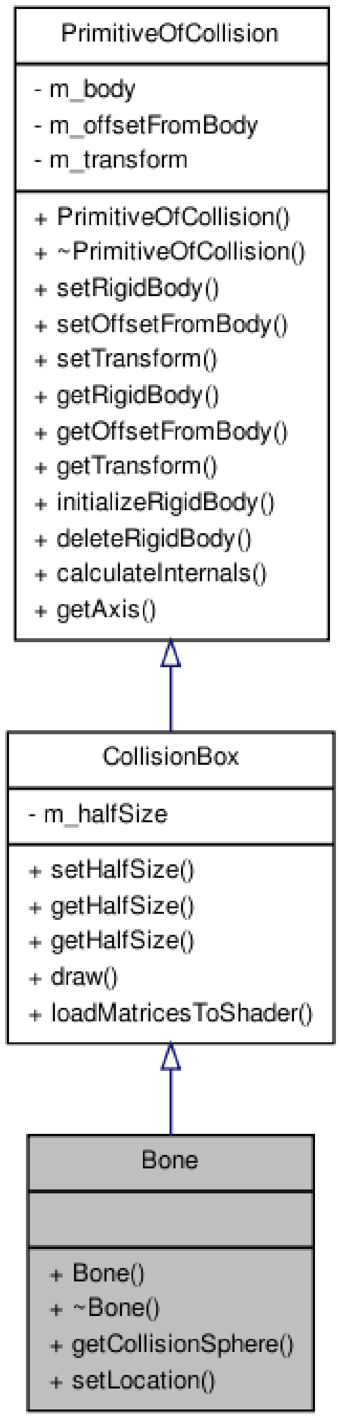
6.2.1. Collision Resolution

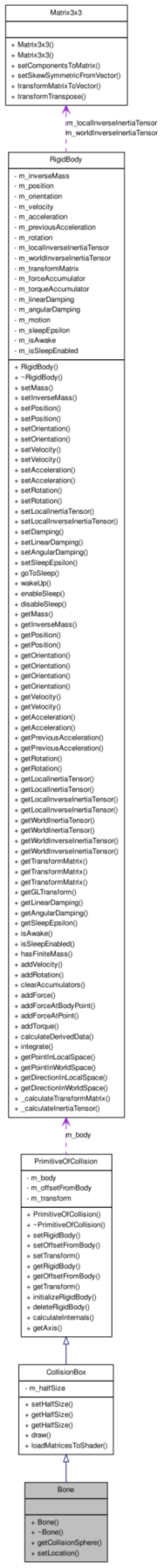


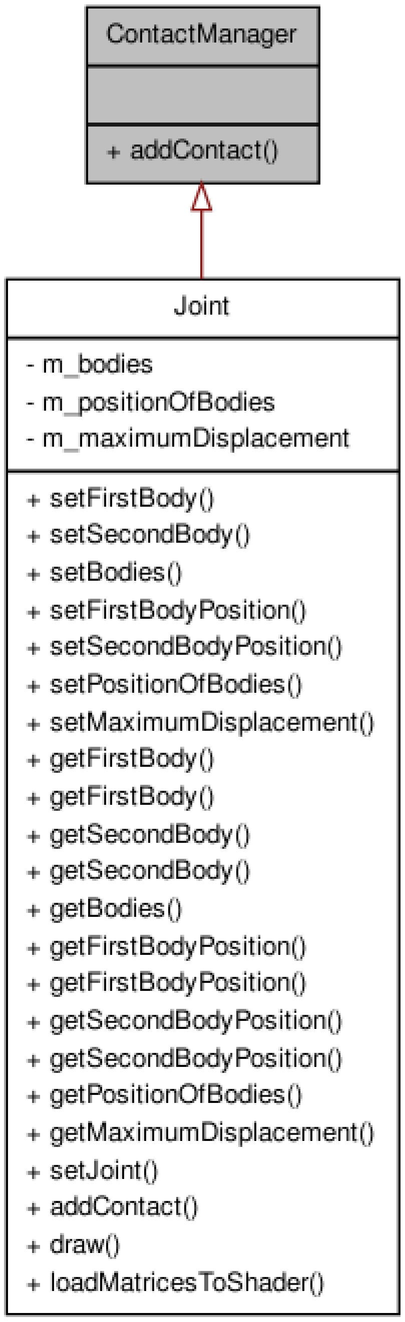



ContactManager.h
Joint.h
Joint.cpp
Screen.h
Screen.cpp
MainWindow.h
main.cpp
MainWindow.cpp

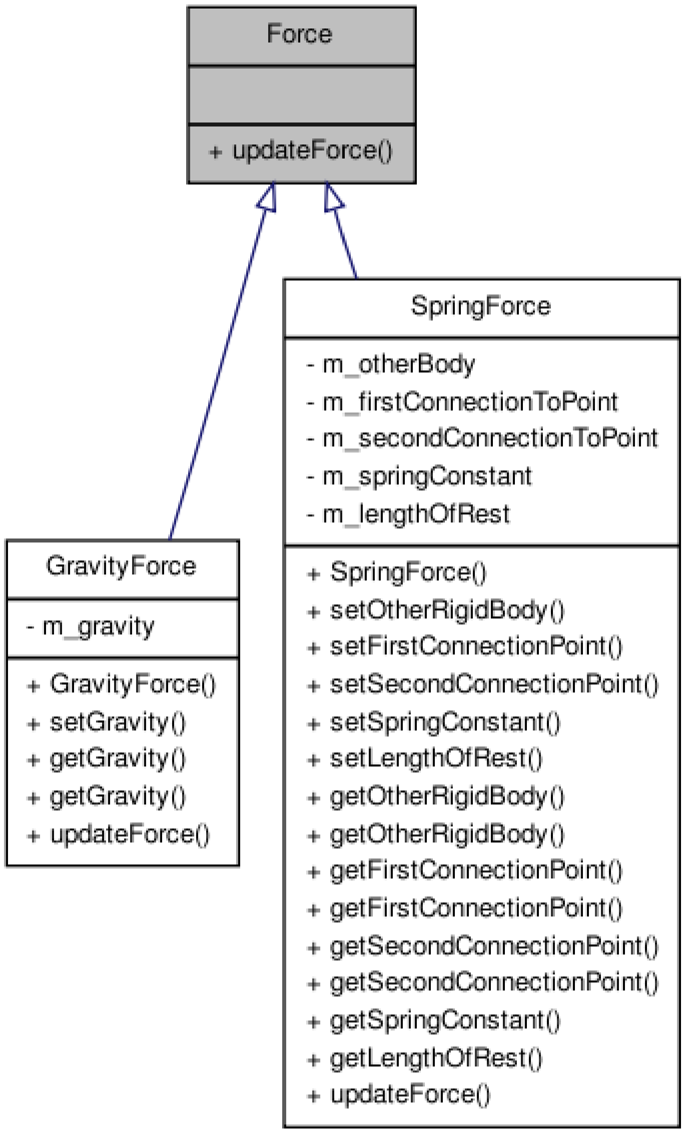

Joint.h
Joint.cpp
Screen.h
Screen.cpp
MainWindow.h
main.cpp
MainWindow.cpp


PrimitiveOfCollision
- m_body
- m_offsetFromBody
- m_transform
+ PrimitiveOfCollision()
+ ~PrimitiveOfCollision()
+ setRigidBody()
+ setOffsetFromBody()
+ setTransform()
+ getRigidBody()
+ getOffsetFromBody()
+ getTransform()
+ initializeRigidBody()
+ deleteRigidBody()
+ calculateInternals()
+ getAxis()
CollisionBox
- m_halfSize
+ setHalfSize()
+ getHalfSize()
+ getHalfSize()
+ draw()
+ loadMatricesToShader()
Bone
+ Bone()
+ ~Bone()
+ getCollisionSphere()
+ setLocation()


Matrix3x3
+ Matrix3x3()
+ Matrix3x3()
+ setComponentsToMatrix()
+ setSkewSymmetricFromVector()
+ transformMatrixToVector()
+ transformTranspose()
m_localInverseInertiaTensor
m_worldInverseInertiaTensor
RigidBody
- m_inverseMass
- m_position
- m_orientation
- m_velocity
- m_acceleration
- m_previousAcceleration
- m_rotation
- m_localInverseInertiaTensor
- m_worldInverseInertiaTensor
- m_transformMatrix
- m_forceAccumulator
- m_torqueAccumulator
- m_linearDamping
- m_angularDamping
- m_motion
- m_sleepEpsilon
- m_isAwake
- m_isSleepEnabled
+ RigidBody()
+ ~RigidBody()
+ setMass()
+ setInverseMass()
+ setPosition()
+ setPosition()
+ setOrientation()
+ setOrientation()
+ setVelocity()
+ setVelocity()
+ setAcceleration()
+ setAcceleration()
+ setRotation()
+ setRotation()
+ setLocalInertiaTensor()
+ setLocalInverseInertiaTensor()
+ setDamping()
+ setLinearDamping()
+ setAngularDamping()
+ setSleepEpsilon()
+ goToSleep()
+ wakeUp()
+ enableSleep()
+ disableSleep()
+ getMass()
+ getInverseMass()
+ getPosition()
+ getPosition()
+ getOrientation()
+ getOrientation()
+ getOrientation()
+ getOrientation()
+ getVelocity()
+ getVelocity()
+ getAcceleration()
+ getAcceleration()
+ getPreviousAcceleration()
+ getPreviousAcceleration()
+ getRotation()
+ getRotation()
+ getLocalInertiaTensor()
+ getLocalInertiaTensor()
+ getLocalInverseInertiaTensor()
+ getLocalInverseInertiaTensor()
+ getWorldInertiaTensor()
+ getWorldInertiaTensor()
+ getWorldInverseInertiaTensor()
+ getWorldInverseInertiaTensor()
+ getTransformMatrix()
+ getTransformMatrix()
+ getTransformMatrix()
+ getGLTransform()
+ getLinearDamping()
+ getAngularDamping()
+ getSleepEpsilon()
+ isAwake()
+ isSleepEnabled()
+ hasFiniteMass()
+ addVelocity()
+ addRotation()
+ clearAccumulators()
+ addForce()
+ addForceAtBodyPoint()
+ addForceAtPoint()
+ addTorque()
+ calculateDerivedData()
+ integrate()
+ getPointInLocalSpace()
+ getPointInWorldSpace()
+ getDirectionInLocalSpace()
+ getDirectionInWorldSpace()
+ _calculateTransformMatrix()
+ _calculateInertiaTensor()
m_body
PrimitiveOfCollision
- m_body
- m_offsetFromBody
- m_transform
+ PrimitiveOfCollision()
+ ~PrimitiveOfCollision()
+ setRigidBody()
+ setOffsetFromBody()
+ setTransform()
+ getRigidBody()
+ getOffsetFromBody()
+ getTransform()
+ initializeRigidBody()
+ deleteRigidBody()
+ calculateInternals()
+ getAxis()
CollisionBox
- m_halfSize
+ setHalfSize()
+ getHalfSize()
+ getHalfSize()
+ draw()
+ loadMatricesToShader()
Bone
+ Bone()
+ ~Bone()
+ getCollisionSphere()
+ setLocation()


ContactManager
+ addContact()
Joint
- m_bodies
- m_positionOfBodies
- m_maximumDisplacement
+ setFirstBody()
+ setSecondBody()
+ setBodies()
+ setFirstBodyPosition()
+ setSecondBodyPosition()
+ setPositionOfBodies()
+ setMaximumDisplacement()
+ getFirstBody()
+ getFirstBody()
+ getSecondBody()
+ getSecondBody()
+ getBodies()
+ getFirstBodyPosition()
+ getFirstBodyPosition()
+ getSecondBodyPosition()
+ getSecondBodyPosition()
+ getPositionOfBodies()
+ getMaximumDisplacement()
+ setJoint()
+ addContact()
+ draw()
+ loadMatricesToShader()

6.2.2. *Force Management*



7. Conclusion

Ragdoll dynamics is a useful tool which is evolving and constantly being improved to have a better implementation with time and effort from researchers and interested parties who find new and more efficient ways of developing physical simulations for computer generated imagery.
Even though this thesis made a very basic overview of ragdoll simulations and the necessary tools that enable it to work correctly and give an appropriate result. There is still room for improvements on the topic and as it is a tool that is widely used in the game industry, it is also a desirable topic for aspiring game developers to be familiar with.
While there is still a lot of work to be done in the implementation applied in this project it was a good experience and brought a better understanding of the way physics work in games and more specifically in rigid body simulations.

7.1. Limitations

A correct rigid body implementation is very difficult to achieve without a proper understanding of every aspect of the process, which requires a lot of research and testing to search for the best approach to resolve problems that might arise in the process. This is the main problematic aspect in the proper implementation of ragdoll dynamics, because one small error in calculation might cause a drawback in the process and be held back solving the problems that arise instead of progressing with the application.
Also, constraints are a very delicate subject that needs a lot of validations and testing for them to be properly implemented and for the ragdoll simulation to have an appropriate and more natural movement.

7.2. Future work

This application would definitely benefit of more testing and improvements to make it work properly. The impulse calculations still need some tweaking and the ball and hinge joint constraints still need to be included to correctly restrict the body's movement during collision.

It would also be beneficial to the project to attempt to improve it by adding an artificial intelligence to help it make more accurate decisions and have a more natural motion. By doing this, it could be extended into other branches of physics for a more complete scenario.

Another possible implementation could be to have an alternative method by applying an inverse pendulum to the body to help it maintain a straight position and react to the environment standing as opposed to dropping and looking like a lifeless body.

Bibliography

Arikan, O., Forsyth, D. A. & O'Brien, J. F., 2005. *Pushing People Around.* New York, ACM SIGGRAPH.

Dunn, F. & Parberry, I., 2011. *3D Math Primer for Graphics and Game Development.* Second ed. Boca Raton(Florida): A K Peters/CRC Press.

Garstenauer, H., 2006. *A Unified Framework for Rigid Body Dynamics,* Linz, Austria: Johannes Kepler University Linz.

Glimberg, S. & Engel, M., 2007. *Comparison of ragdoll methods - Physics-based animation,* Copenhagen: s.n.

Hall, C., 2003. *AOE 4140 Spacecraft Dynamics and Control.* [Online]
Available at: http://www.dept.aoe.vt.edu/~cdhall/courses/aoe4140/rigid.pdf
[Accessed July 2012].

Hecker, C., 1996. *Physics, The Next Frontier.* [Online]
Available at: http://chrishecker.com/Rigid_body_dynamics
[Accessed August 2012].

Hecker, C., 1997. *Physics, Part 2: Angular Effects.* [Online]
Available at: http://chrishecker.com/Rigid_body_dynamics
[Accessed August 2012].

Hecker, C., 1997. *Physics, Part 3: Collision Response.* [Online]
Available at: http://chrishecker.com/Rigid_body_dynamics
[Accessed August 2012].

Hecker, C., 1997. *Physics, Part 4: The Third Dimension.* [Online]
Available at: http://chrishecker.com/Rigid_body_dynamics
[Accessed August 2012].

Hennix, M. et al., 2003. *Rag doll physics,* Norrkoping, Sweeden: Scandinavian Simulation Society.

Kenwright, B., 2012. *Game Physics Programming...* [Online]
Available at: http://www.xbdev.net/physics/ImpulseDemo/index.php

Millington, I., 2007. *Game Physics Engine Development.* San Francisco(California): Morgan Kaufmann.