Real-time Vehicle Simulation for Video Games
Using the Bullet Physics Library

Hammad Mazhar
Contributors: Dan Negrut

Simulation Based Engineering Lab
University of Wisconsin Madison

September 17, 2008
Abstract

This report presents a detailed walkthrough on how a racing game was created using the Bullet Physics Library. This document covers the developmental process of the game and contains a detailed explanation of parameters pertaining to the vehicle model. A complete version history is presented that includes early design prototypes, failures and improvements that were made to the game. Also presented here are details about the Heads up Display (HUD), the Windows Forms Graphical User Interface (GUI), and any OpenGL optimizations and effects used.

Contents

1. Introduction ................................................................................................................. 2
2. Novint Falcon .............................................................................................................. 2
3. Bullet Physics Library ................................................................................................. 3
4. Vehicle Model ............................................................................................................. 3
   4.1 Specifications and Constants .............................................................................. 3
   4.2 Bullet Implementation ........................................................................................ 4
      4.2.1 Drawbacks to the Model ............................................................................... 4
   4.3 Description of Parameters ................................................................................... 5
   4.4 Implementation of Parameters ............................................................................ 5
      4.4.1 Engine Torque ............................................................................................... 5
      4.4.2 Gearbox ......................................................................................................... 5
      4.4.3 Suspension .................................................................................................... 6
      4.4.4 Aerodynamic Drag ........................................................................................ 8
      4.4.5 Downforce ..................................................................................................... 8
      4.4.6 Wheel Friction .............................................................................................. 8
5. Track Mesh ................................................................................................................. 9
   5.1 Collision mesh .................................................................................................... 9
   5.2 Visualization mesh .............................................................................................. 9
6. User Interface ............................................................................................................ 10
   6.1 Windows Forms ................................................................................................ 10
   6.2 OpenGL HUD ................................................................................................... 10
7. OpenGL ..................................................................................................................... 11
   7.1 Optimizations .................................................................................................... 11
   7.2 Effects ............................................................................................................... 11
8. Game creation ........................................................................................................... 12
   8.1 Version 1 ........................................................................................................... 12
   8.2 Version 2 ........................................................................................................... 13
   8.3 Version 3 ........................................................................................................... 13
   8.4 Version 4 ........................................................................................................... 13
   8.5 Version 5 ........................................................................................................... 14
9. Conclusion ................................................................................................................ 16
10. Acknowledgements ................................................................................................... 17
11. Reference .................................................................................................................. 18
1. Introduction

This racing game was created for the ProCSI [1] outreach program - 2008. This game was used during the hands on portion of one of the modules presented to the students. The students competed against each other for the fastest lap time. The main purpose of this game, other than being a ProCSI module, was to explore the applicability of the Novint Falcon [2] with the Bullet Physics Library [3]. Physical feedback was one of the first features introduced that matured with each version of the game. Throughout these version changes the game evolved, many new game play and visual elements were introduced and expanded upon. One important design rule that was adopted early on was to make sure that the game was as true to life as possible. Once the foundation of the game was laid out, a real vehicle was chosen to be modeled. In this case the Audi R8’s actual parameters were used so that a realistic approximation could be created further immersing the player into the game.

2. Novint Falcon

The Novint Falcon [2] is a haptic device, which provides physical feedback to in-game physics. The Falcon was used to provide an alternate method for input. The throttle was controlled with Z axis, steering was controlled with X axis and feedback was applied using the amount of G’s acting upon the vehicle. This created resistance that made it more difficult for the player to turn and accelerate the vehicle. For example, if the player steers the vehicle to the right and the vehicle was experiencing G’s towards the left, the Falcon would create a feedback force towards the left.
3. Bullet Physics Library

The Bullet Physics Library [3] is a free, open source cross platform C++ SDK geared towards the creation of games and other visual presentations. The library is not natively OpenGL and can be used in conjunction with other 3D API’s such as DirectX. OpenGL was used for this game because it is more widely supported and works on different operating systems also many of the example programs that come with Bullet use the OpenGL API to visualize the simulation. Bullet version 1.69 was used for this game.

4. Vehicle Model

The vehicle model chosen for this game was based on the Audi R8. The vehicle mesh [4] shown in Figure 1 was the 3D model used in-game.

![Vehicle Mesh](image)

4.1 Specifications and Constants

Specifications for the 2008 Audi R8:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>4.42976 m</td>
</tr>
<tr>
<td>Width</td>
<td>2.02946 m</td>
</tr>
<tr>
<td>Height</td>
<td>1.25222 m</td>
</tr>
<tr>
<td>Weight</td>
<td>1564.89 kg</td>
</tr>
<tr>
<td>Horse Power</td>
<td>420 HP SAE @ 8,250 RPM</td>
</tr>
<tr>
<td>Torque</td>
<td>430 Nm @ 4,500 RPM</td>
</tr>
<tr>
<td>Gear Ratios:</td>
<td></td>
</tr>
<tr>
<td>First</td>
<td>4.37:1</td>
</tr>
<tr>
<td>Second</td>
<td>2.71:1</td>
</tr>
<tr>
<td>Third</td>
<td>1.88:1</td>
</tr>
<tr>
<td>Fourth</td>
<td>1.41:1</td>
</tr>
<tr>
<td>Fifth</td>
<td>1.13:1</td>
</tr>
<tr>
<td>Sixth</td>
<td>0.93:1</td>
</tr>
<tr>
<td>Final Drive (Axle Ratio)</td>
<td>3.46:1</td>
</tr>
<tr>
<td>Reverse</td>
<td>3.71:1</td>
</tr>
<tr>
<td>Wheels (Radius/Width in meters):</td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td>Front</td>
<td>.4572/.2159</td>
</tr>
<tr>
<td>Rear</td>
<td>.4572/.2667</td>
</tr>
<tr>
<td>Drag Coefficient [7] (Cx)</td>
<td>.340</td>
</tr>
<tr>
<td>Downforce coefficient (Cd)</td>
<td>.4</td>
</tr>
<tr>
<td>(estimated)</td>
<td></td>
</tr>
<tr>
<td>Frontal Area [8] (Fa)</td>
<td>1.99 m²</td>
</tr>
<tr>
<td>Other Constants:</td>
<td></td>
</tr>
<tr>
<td>Air density (ρ) at 20°C at sea level</td>
<td>1.2041 kg/m³</td>
</tr>
</tbody>
</table>

**Table 1: Audi R8 Specifications used to model the vehicle**

4.2 Bullet Implementation

Basic vehicle implementation for this game was initially straightforward because the Bullet Physics Library includes a simple vehicle model. This model, called a **btRaycastVehicle**, figure 6, supports some basic chassis, wheel and suspension parameters. This model greatly reduces the time needed to create a racing game by providing the basic functions and parameters needed to control a vehicle.

4.2.1 Drawbacks to the Model

This model had several shortcomings that needed to be addressed:

- Force applied to Chassis rather than Wheels
- No Engine Implementation
  - Vehicle has constant indefinite acceleration
  - Gears not implemented
- Vehicle became unstable as velocity increased

The chassis was driven by a linear force; this was the greatest limitation to this model. Normally the chassis of a vehicle moves because its wheels are rotating and in contact with this ground. Because the chassis was being driven by a linear force, the wheels acted as though they were rotating because the chassis was moving. This had two major effects, the first of which was that the wheels could not spin or lose traction if too much force (or torque) was applied to them. Second, proper drift characteristics could not be introduced into the game. Drifting is a driving technique where the rear wheels intentionally lose grip and skid through a turn. This swings the rear of the vehicle around the turn and allows the vehicle to maintain its velocity. The drifting effect added realism to high speed turns by allowing the vehicle to skid. The absence of a sophisticated engine model was another limitation; the only parameter that could be controlled was the engine force. This parameter was either on or off meaning that the vehicle, as long as it was under control, had no upper limit to its velocity. To fix this, a gearbox, engine RPM and torque needed to be implemented so that the vehicle velocity could be properly controlled. Finally, the basic vehicle setup was inherently unstable if the vehicle was given realistic dimensions, height and wheel size. A disproportionately wide vehicle could handle faster corners better than narrow vehicles but would still roll over easily. Changes needed to be made so that the vehicle could be dimensionally equivalent to the R8 and remain stable. The
parameters in section 3.3 were either added or controlled in order to create a realistic, more stable approximation of the vehicle.

4.3 Description of Parameters

The following parameters were used to create the vehicle model for the game. Suspension was the only parameter not implemented from the ground up because the Bullet vehicle model included a suspension model sufficient for this game.

- **Engine Torque**: The amount of torque exerted on each wheel from the engine.
- **Gearbox**: Defines the gear ratios and the RPM’s at which the gears shift.
- **Suspension**: Defines the handling and ride characteristics.
- **Aerodynamic drag**: Force produced by air resisting the vehicle as it travels.
- **Downforce**: Amount of force pushing the vehicle into the road.
- **Wheel Friction**: Defines the slip and traction characteristics of the tires.

4.4 Implementation of Parameters

The following are details about implementation and usage of the parameters described in section 3.3

4.4.1 Engine Torque

Engine torque was the magnitude of force created by the engine that turns the crankshaft. Equation 1 shows the formula used to determine the torque produced by the engine.

\[
\text{Torque (N/m)} = 1.000003 \cdot \text{RPM}^2 + 0.04 \cdot \text{RPM} + 300
\]  

(1)

While the torque curve defined by this equation was not entirely realistic, it provided adequate realism during acceleration and allowed the vehicle to match the actual 0-60 MPH time of the Audi R8. Using equation 2, the calculated torque was then converted into a linear force, which was subsequently applied to the wheels through the vehicle model’s EngineForce parameter. This equation allowed the existing linear force model to be interfaced with the new engine model.

\[
\text{Force (N)} = \frac{\text{Torque}}{\text{Wheel radius}}
\]  

(2)

4.4.2 Gearbox

The gearbox was modeled with the following conditional statements:

- If RPM is greater than 8000, shift up.
- If RPM is less than 1000 times the current gear ratio, shift down.
- If in sixth gear, RPM cannot exceed 8250 RPM.
- RPM increases by a fixed amount while accelerating depending on the gear.
- RPM slowly decrease if the vehicle is not accelerating
- Engine force is zero when gear is changing.
These simple conditions allowed the vehicle to shift up and down in a realistic manner. Once the vehicle shifts up, the RPM was decreased to the amount required to maintain the same velocity for the new gear. Likewise, if the vehicle downshifts, the RPM will increase. The third condition forced the vehicle to reach a maximum velocity and prevented it from accelerating further. When gears shift the engine force goes to zero, jerking the vehicle forwards in a realistic manner. The RPM of the engine decreases over ~1 second before completing the gear change and resuming acceleration.

In an effort to create realistic reversing, several additional conditions were stated:

- The vehicle cannot reverse if the forward velocity is greater than ~1-2 mph.
- The vehicle cannot reverse if its gear is greater than 1.
- If the vehicle is traveling at a velocity greater than ~1-2 mph the brake is applied until the velocity has decreased to the allowable amount.

These conditions properly simulated a vehicle in reverse and allowed for braking to be modeled.

### 4.4.3 Suspension

The suspension model for the vehicle was largely unchanged throughout the making of this game. The suspension defined in the vehicle model included many parameters that could be adjusted to change the vehicle’s behavior. Some of these parameters could be modified by the player during the game. The main parameters are as follows:

- Suspension Stiffness
- Suspension Damping
- Suspension Compression
- Roll Influence
- Suspension Free Length
Suspension rest length and stiffness were the most influential vehicle parameters in terms of stability and handling. By controlling the suspension rest length the vehicle could be lowered to or raised from the ground (Figure 2). Theoretically, there was no upper limit to the rest length but a bounding value was chosen. If no such bounding value was implemented, the vehicle could become higher than the boundary wall causing it to fall off of the side of the track. Good handling was achieved if the vehicle was low to the ground rather than higher up as the vehicle would tip over easily. If the suspension stiffness was too low the vehicle would drag on the ground, even if the suspension height was large. If the stiffness was too great the vehicle would likely roll over during a fast corner. This was due to the stiffness being so large that the suspension couldn’t travel enough to allow the vehicle to lean during a turn. Suspension compression and roll influence remained constant throughout the game. Roll influence controlled the vehicle’s tendency to roll over if the center of mass moves above the wheels. This parameter was set to 1.0 because it reduced the vehicles tendency to roll over. Suspension compression controls the speed with which the suspension travels and was set to 1.0 so that the vehicle cornered smoothly. Low suspension travel along with high suspension damping help to effectively damp a vehicle’s suspension. Suspension dampening was another important factor in determining the vehicle’s cornering ability. If the damping value was too low the vehicle would continually oscillate in all directions. As the damping value was increased the amount of shock that the suspension can handle without transferring it to the vehicle also increased. If the damping value was too high the suspension would
dampen itself because it wasn’t strong enough to overcome the resistive force of the damping. In the real world it would be akin to using a very thick fluid inside of the suspension.

### 4.4.4 Aerodynamic Drag

Aerodynamic drag or air resistance acts upon a vehicle by resisting its motion. The more aerodynamic a vehicle is, the less drag/resistance it experiences. This parameter was implemented because, just as in real life, the engine can never reach its full potential due to resistive forces acting upon it. In this game aerodynamic drag was modeled by a force in front of the vehicle pushing back on it. Equation 3 was used to calculate the force induced by drag. [9]

\[
F_d = \frac{1}{2} \cdot C_x \cdot F_a \cdot \rho \cdot v^2
\]  

\(F_d\) (Newtons) = \(1/2 \cdot C_x \cdot F_a \cdot \rho \cdot v^2\) \hspace{1cm} (3)

\(C_x\), frontal area \((F_a)\), and the air density \((\rho)\) were predetermined values. The \(C_x\), or drag coefficient, could be modified by the player during the game. If the player lowers the \(C_x\) then a higher maximum velocity was achievable.

### 4.4.5 Downforce

Downforce acts upon a vehicle by applying a downward force making it more stable and handled better due to the increased traction in the tires. In this game, downforce was applied to the vehicle by creating a force above the vehicle that pointed downwards. Equation 4 was used to determine the magnitude of downforce produced by the vehicle. [9]

\[
\text{Downforce (Newtons)} = \frac{1}{2} \cdot C_d \cdot F_a \cdot \rho \cdot v^2
\]

\(\text{Downforce (Newtons)} = \frac{1}{2} \cdot C_d \cdot F_a \cdot \rho \cdot v^2\) \hspace{1cm} (4)

In this equation only the downforce coefficient \((C_d)\), frontal area \((F_a)\), and air density \((\rho)\) are predetermined values. The downforce coefficient could be modified by the player during the game. A higher coefficient created a greater force above the vehicle thus making it more stable. Other than stability and decreased body roll through high speed turns, increasing the downforce had no noticeable effects. Normally a vehicle experiences increased grip in the tires but no such effect was present because wheel friction was explicitly controlled.

### 4.4.6 Wheel Friction

Wheel friction was modeled as a step function. The friction was constant until the vehicle experienced a lateral acceleration greater than a certain amount. Once the lateral acceleration was greater than the bounding value, the friction coefficient decreased to a lower value. If the lateral acceleration decreases below the bounding value, the friction coefficient increased to its original value. There was one main consequence to using this approach, which was that the friction coefficient rapidly oscillated between slipping and not slipping. If the vehicle begins to slip (because of its high lateral velocity) the wheel friction decreases, the vehicle then loses traction, causing the lateral velocity to decrease. Immediately the friction coefficient increases causing the lateral velocity to increase.
above the bounding value and force the wheels to lose traction and slip again. The speed at which the friction oscillated was dependent on the lateral velocity of the vehicle; higher lateral velocities resulted in the vehicle spending more time sliding while lower velocities caused the vehicle to spend more time in traction. Consequently, this allowed the vehicle to drift around corners if the friction coefficient on the rear wheels is lower than on the front. The rear of the vehicle would swing out as the car slid, allowing a drift to be completed. The process by which the vehicle enters and exits the drift is as follows: the vehicle enters a turn at high velocity causing slipping to occur. At the start of the turn the fast oscillations of the friction coefficient causes the vehicle to slide, at this point more time is spent sliding than not. Once the vehicle is sliding it begins to lose velocity, after a certain point more time is spent in traction than slipping. Once this occurs the vehicle regains control, allowing the player to exit the drift in a controlled manner.

5. Track Mesh

In this game the track mesh consisted of two elements, the collision mesh and the visualization mesh. This separation allowed each mesh to be optimized, loaded and used separately in the game. The collision mesh could remain relatively simple while greater detail was displayed onto the screen using the visualization mesh. The collision mesh was created in Blender [10], a free 3D modeling program, and the visualization mesh was created in a modeling program called 3ds Max. [11]

5.1 Collision mesh

The collision mesh used in this game consisted of a closed winding track with raised walls. The track was created in Blender by modeling a closed spline which was then swept to give it width. The inner and outer edges of the road were then extruded upwards by an amount several times greater than the height of the vehicle. This created a wall and prevented the vehicle from inadvertently leaving the track. The collision mesh was then exported out of blender as a Collada physics file using the Blender-Collada exporter. The file is then loaded into the game using the Bullet/Collada library. The Collada physics format[12] is a standard format widely used in industry and is supported across many programs and libraries.

5.2 Visualization mesh

The visualization mesh consisted of a closed track with a city-like structure filling the inside perimeter. The mesh was created by exporting the collision mesh from Blender as a 3D model in wavefront (obj) format. This file was further modified in 3ds max. The walls of the track were lowered and the track was colored. The city model was created by using the Greeble 3ds max plug-in. This free plug-in can create random building like objects on top of polygons. This plug-in simplified the creation of a complex city into a few steps. Figure 3 shows what a basic plane looks like after the Greeble plug-in was applied to it.
Figure 4: Greeble Plug-in in 3ds max creates a city-scape from a plane with polygons

6. User Interface

6.1 Windows Forms

Windows Forms offer a simple way of creating an application with a Graphical User Interface (GUI). They are guaranteed to work on any Windows machine provided that the Microsoft .Net Runtime is installed. In order to use Windows forms with OpenGL a Windows form object called a panel needs to be subclassed so that proper contexts can be provided to the OpenGL window. In order to use the panel, it needs to be initialized at the start of the program. Any draw call made from the render function of the panel will be shown on the screen.

6.2 OpenGL HUD

The Heads Up Display (HUD) was used to display information to the player. It was created by drawing an orthographic projection on top of the 3D elements in the game. Basic OpenGL calls were used to draw text, lines, points, and polygons onto the screen. Some of the elements shown on the HUD are as follows:

- Position on track
- Percent track completed.
- Velocity
- RPM
- Current gear
- Distance traveled
- Lap number
- Best lap time
- Current lap time
- Frame rate

Text was created by using a Windows specific function that created display lists for a specified font. A font could be specified to the function which would then be loaded at runtime. The font could be loaded from the game’s local directory so that the player’s computer didn’t need the font in order for the font to be loaded.
7. OpenGL

7.1 Optimizations

Many optimizations were made to this game so that it could run on a low power machine. Before optimizations, this game ran at 60 fps with approximately 700 to 800 thousand polygons on the screen at a time. An Nvidia 8800 GT is able to draw this many polygons but lower powered video cards ran at around 7 fps with the same settings. The first optimization made was to lower and fix the maximum frame rate at 30fps. This fixed frame rate inconsistencies such as the frame rate noticeably increasing when fewer polygons were displayed on the screen. The second optimization was to reduce the maximum number of polygons that could be displayed onto the screen. This was a multi-step process that entailed the removal and reduction of polygons. Figure 5 shows the polygon reduction before parts of the vehicle were removed. Polygons that could not be seen in certain views were completely removed. For the chase view the camera was behind the vehicle so the front grill and lights were removed. Heavy reductions were made to objects occluded by glass and moderate reductions were made to visible objects. For the inside view, because most of the vehicle was not visible to the player, only small portions of the hood, dash and windshield were kept. For the bumper cam, the entire vehicle was hidden. The city mesh was also optimized by using larger buildings with a lower number of polygons.

Figure 5: Mesh Optimization, mesh on left is original and mesh on right is optimized

The third step was to remove any textures that were present on the vehicle mesh. This reduced memory usage for the vehicle mesh on the vehicle. The fourth and last optimization used OpenGL keywords to reduce the polygon, point and line smoothing quality. These optimizations allowed the game to be run on older hardware with constant frame rates. A large improvement was seen from before optimizing, with a frame rate of 7, to after optimizing, with a constant frame rate of 30.

7.2 Effects

Several effects were incorporated but, due to frame rate restriction, some did not make it into the final version of the game. In the final version the effects included:
shadows, point line smoothing, and transparency. Some effects that were removed or disabled included a rear view mirror on the HUD and inside the vehicle, along with fog effects. Shadows were used in the opening scene of the game.

8. Game creation

This game began as a basic demonstration of what the Novint Falcon could accomplish with the vehicle model in Bullet. By the final version of the game, physical feedback was still integrated into the game but more emphasis was placed on realism.

8.1 Version 1

The first version of the game, shown in figure 6, drew upon the vehicle demo included in Bullet and added Novint Falcon support to it. Some rigid boxes were added so that the user could feel the collisions as they occurred. A mapping of the change in vehicle velocity was used as the force that the user experienced. This mapping provided adequate, although not realistic, feedback to the user. This version was also used to test the effects of different vehicle parameters included with the model. Windows forms were also used in this game although not all of the available controls were implemented.

Figure 6: First Version of the game, no vehicle model was present at this point
8.2 Version 2

Version two added an environment to the game in the form of a height field. A height field consists of a two dimensional grayscale image (a height map) whose values for each pixel are mapped onto a grid of polygons of the exact same size. Bullet contains an object called a btHeightField which can load height maps in the .RAW format. This height field was used to define the shape of the ground. The height map was created in Gimp [13] and consisted of varying shades of gray. A road texture was also created in Gimp and placed on top of the height field. In order to add more detail to the ground, a larger height field was created. One noticeable effect of a larger height field was a decrease in frame rate. As more polygons were displayed on the screen, the video card was not able to draw all the polygons at once. This problem was traced back to the draw function for the height field; no optimizations were implemented and so the full potential of the video card could not be used. Because of this problem, walls could not be created around the perimeter of the racetrack. If a clean, straight wall was needed, then the resolution of the height field would need to be several times larger and this was not possible. Height fields were ultimately abandoned for Collada meshes.

8.3 Version 3

Version three improved upon the performance and visuals of the game by leveraging the Collada physics format. These meshes allowed for arbitrary shapes to be imported into Bullet with preset physical parameters. Collada meshes were created by modeling ordinary geometry in Blender, applying simulation parameters (static/dynamic etc.) and then exporting them using the Collada file exporter. The mesh was loaded into the game by using the Bullet Collada import class; all of the parameters set in Blender were automatically applied. While this format greatly improved the playability, visuals and performance of the game, a new problem was discovered. If the mesh was too large, parts would get cut off when it was drawn. The vehicle continued to drive on the track but the track could not be seen underneath the vehicle. In order to solve this problem, the lib3ds library was used to load .3ds models and use them in conjunction with the Collada files.

8.4 Version 4

Version four introduced the greatest improvements in visuals and performance. By using Lib3ds, the collision mesh could be separated from the Visualization mesh, what you saw didn’t necessarily have to be simulated. This approach required that the track be loaded twice, once as a Collada mesh and once as a .3ds mesh. To get the .3ds mesh, the collision mesh of the track was exported as an .obj file from Blender and imported into 3ds max. The track was colored, its walls were lowered and a city model was created and placed in the center of the track. Then the file was exported as a .3ds file and loaded into the game. Because the collision mesh was copied to the visualization mesh and was not translated or rotated, the visuals loaded exactly on top of the collision mesh. This ensured that collisions with the walls of the track looked real. A similar technique was used to place the vehicle atop its representation in the game. The Audi R8 model was first split into two parts in 3ds max (figure 7). The wheels were separated from the chassis, all but one wheel was removed, and the chassis and wheels were placed at the origin.
An Audi R8 was used as the replacement; this model was separated into a wheel model and a chassis model. The wheel model was loaded for all of the wheels and the chassis was loaded in place for the box. The models needed to be rotated and scaled in order to fit and rotate properly with the vehicle (figure 8).

8.5 Version 5

The fifth version of the game, figure 9, saw vast improvements in playability and game mechanics. A track model was introduced along with a HUD, GUI, engine and gearbox.

In order to display track completion data, lap number and current track position, a track model was needed. To determine the current track position and direction, a set of points were created around the track. Using the Track mesh data from 3ds max a list of
1000 points was created; these points were in the center of the track and completed one full circuit. By measuring the distance to each point, the closest point could be determined. This point was the current position of the vehicle, because the points were sequential (from 1-1000) the direction of travel was known. If the vehicle’s current position number was increasing, it was heading in the right direction. Otherwise, if the numbers were decreasing the vehicle was traveling in the wrong direction and a “Wrong Way” message would be displayed to the player.

The vehicle rollover issue was also corrected in this version. By lowering the center of mass and adjusting the wheel to chassis joints, the vehicle became slightly more stable. This did not completely eliminate vehicle rollover; it was discovered that if the wheel radius in the simulation was reduced, the stability noticeably increased. While this was a less than optimal solution it could not be noticed by the player because the visual representation of the wheels remained the same.

Using the Windows Forms designer integrated with Visual Studio, a User Interface (UI) was designed. The UI featured several sliders, labels, drop downs, checkboxes and one button which reset the vehicles position and orientation. Different event handlers were created for each input/UI object depending on the type of event triggered by that object. For example a checkbox emits a CheckStateChanged event while a timer emits a Tick event. A separate function, or event handler, was created for that object to process data and change parameters within the game. In order for the user to modify parameters during the game, the game would pause if the player moved their mouse over the bottom bar which contained the UI. Windows Forms was also used to time the game and keep frame rates consistent. Two timer objects were created, the first emitted an event every 20 milliseconds, the second every 1000 milliseconds. The event handler for the first timer event called the ‘draw’ function and iterated one time step. The second timer was used to calculate the frame rate; to do this an integer variable was created which was incremented every draw call. When this timer activated after one second, the value of the variable was the number of draw calls per second (frame rate), the variable was then reset to begin counting again. The second timer also kept track of lap times; the time was stored and the best lap time was updated if it was faster than the previous time.

After the engine and gearbox were implemented, there was enough data to begin work on the HUD. Several accessing functions were created which returned the data that needed to be displayed. sprintf was used to format the data into a C-style string. This array of characters was then passed to the font function which drew the text.
9. Conclusion

This document outlines the process, software and equations used to implement a racing game using Bullet and OpenGL. Bullet is an exceptional C++ physics library that can be leveraged for a variety of video game and visualization applications. While this library is not meant for mathematically accurate physics, it can easily achieve real time performance for video games where realistic accuracy is not needed. In fact, CPU benchmarks showed that the racing game only used ~5-10% of the CPU even when it was run on older computers. The main bottleneck for this game became the visuals that needed to be drawn onto the screen. On a current generation video card like the NVIDIA 8800GT the game could run at 60 FPS without any optimizations. This was not the case for older, slower, video cards, many optimizations were implemented and features removed so that a stable frame rate of 30 FPS could be achieved.

Through this project a lot was learned about testing and debugging on different hardware. This game would run excellently on one computer and not even run on a different one. These types of bugs were difficult to pinpoint and solve, many times a guess and check approach was used to determine what part of code was responsible. Testing was a large part of the developmental process. When new features were added many different test cases needed to be run to make sure that the code was robust. Implementing the gear system is one such example; because there were many conditional statements involved certain scenarios got overlooked. The vehicle might go from first to
second gear but then skip a gear and go to fourth. Testing and checking for these types of problems helped to make the code more stable and user proof code.

This game demonstrated how easy it is to create a semi-accurate racing game on top of the vehicle model that is included in Bullet. While modifications and additions were needed to improve the game, the vehicle and game model only took a few days to implement. Future work on this project deals with new additions to the Bullet library. Soft body support was recently introduced into the library which opened up many new possibilities. Tires could be modeled with soft bodies rather than rigid allowing for a better friction and handling model to be implemented. Future plans also include this game being used in ProCSI’s 2009 session.

10. Acknowledgements

The author would like to acknowledge Dan Negrut, Justin Madsen, Makarand Datar, and Dan Melanz for their support and help with this technical report.
11. Reference


