PB2001-103560

Final Report

CRASH SIMULATION AND ANIMATION
"A NEW APPROACH FOR TRAFFIC SAFETY ANALYSIS"

Dr. Mohamed Abdel-Aty, PE
Center for Advanced Transportation Systems Simulation
University of Central Florida
Orlando, FL 32816
407-823-5657
Fax: 407-823-3315
mabdel@mail.ucf.edu

February 2001

REPRODUCED BY: NTIS
U.S. Department of Commerce
National Technical Information Service
Springfield, Virginia 22161

CRASH SIMULATION AND ANIMATION

"A NEW APPROACH FOR TRAFFIC SAFETY ANALYSIS"

Executive Summary

Fatalities and injuries related to motor vehicle accidents constitute a major societal problem. Statistics show that in 1999 there were an estimated 6,279,000 police reports of traffic accidents. Also in 1999 41,611 people were killed and 3,236,000 were injured in traffic accidents. Injuries from accidents dwarf all other causes of lost human productivity and reduced quality of life. Traffic safety researchers must understand how and why motor vehicle accidents occur in order for them to find treatments and countermeasures. The role of the research presented in this report is to improve this understanding by developing a methodology to animate and simulate traffic accidents. The main source of accident data is the standard accident report forms filled out by the police officers. Some of the information recorded on this form is then coded into computerized format. The traditional methodology to analyze traffic safety is to apply a variety of statistical techniques using these coded accident reports. There are numerous problems with such data. There are many inconsistencies and biases in accident data. Therefore, there is a need to develop a methodology to improve the traditional accident analysis techniques.

This research's objective is to present a methodology to supplement the conventional traffic safety analysis techniques. This methodology aims at using computer simulation to animate and visualize crash occurrence at high-risk locations. This methodology aid into developing the appropriate safety countermeasures for high-risk locations (e.g., intersections, curves, ramps).

A computer accident simulation and animation program has been developed. The program includes many features including accident detection, location, vehicle, and animation attributes. The program has the ability to animate accidents at specific highway location based on the configuration of this location, which is entered by the user. The program is augmented with a database to query for accidents at a certain location by configuration of roadway, and light or weather conditions, then animate them for better understanding of the causes of the accidents. The program is designed to include reconstruction, or the opposite (i.e., given certain circumstances the model can simulate whether an accident will occur). This helps in the efficient design and testing of accident countermeasures.

PROTECTED UNDER INTERNATIONAL COPYRIGHT
ALL RIGHTS RESERVED
NATIONAL TECHNICAL INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE

Reproduced from
best available copy.

Table of Contents

Executive Summary 1

Table of Contents 3

1. Abstract 4

2. Introduction 4

3. Overview of research Objectives and Contribution 8

4. Previous Work 10

5. Limitation of Current Approaches 12

6. The New Approach: The Crash Simulator 13
   - 6.1. Intuitive graphical user interface (GUI) 13
   - 6.2. Simulation Control 14
     - 6.2.1. Intersection Control 14
     - 6.2.2. Car Control 16
     - 6.2.3. View Control Module 17
     - 6.2.4. Action Control Module 21
   - 6.3. Crash Visualization 21
   - 6.4. Vehicle Models 23
   - 6.5. Modularity 23
   - 6.6. Multiplatform Support 24
   - 6.7. Arbitrary Number of Vehicles 24
   - 6.8. Vehicle Path Control 24
   - 6.9. Different Visualization Features 25
   - 6.10. The Database Query Mode 25

7. Implementation 27
   - 7.1. Time Based Simulation 27
   - 7.2. Software Language & Libraries 27
   - 7.3. Physics 28
   - 7.4. Collision Detection 31
   - 7.5. Scenario Generation 35
   - 7.6. Vehicle Models 38

8. Suggestions for the Improvement of the Computer Program 39

9. Conclusions and Recommendations 43

10. References 44

CRASH SIMULATION AND ANIMATION
"A NEW APPROACH FOR TRAFFIC SAFETY ANALYSIS"

1. Abstract

This research's objective is to present a methodology to supplement the conventional traffic safety analysis techniques. This methodology aims at using computer simulation to animate and visualize crash occurrence at high-risk locations. This methodology aids into developing the appropriate safety countermeasures for high-risk locations (e.g., intersections, curves, ramps). A computer accident simulation and animation program has been developed. The program includes many features including accident detection, location, vehicle, and animation attributes. The program has the ability to animate accidents at specific highway location based on the configuration of this location, which is entered by the user.

2. INTRODUCTION

The introduction of the automobile in the early 1900's brought with it little concern for the safety of the occupants of these machines. Early automobiles were slow and few in number and accidents were rarely serious. The years hence have seen a dramatic rise in both the number and performance of automobiles with a resulting increase in the frequency and severity of automobile accidents. Today tens of thousands of people die each year on United States roads and highways.

In 1952, a pioneer program in highway safety research, the Automobile Crash Injury Research program (ACIR), was created with the objective of determining injury causation among occupants of cars involved in accidents, in order that injuries may be prevented or mitigated through improved vehicle design. Shortly after that, in the 1960s, the digital computer was coming of age. These events lead to the National Traffic and Motor Vehicle Safety Act and the National Highway Safety Act signed by President Johnson in 1966. The National Highway Traffic Safety Agency (NHTSA) was born as a result of these acts. NHTSA started to fund research in the area of automobile crash investigation.

Prior to this, those in the business of designing cars had not seriously addressed driving safety and opportunities for improvement were numerous. Safety features easily within the grasp of technology were added to cars as standard equipment and at moderate cost. The addition of seatbelts is an obvious example of an inexpensive, and easily implemented, safety feature with a high payback in lives saved.

Today, with the more easily implemented safety features already included on most cars, making improvements in safety has become more costly, and technically more challenging. The introduction of air bags is an example of such a feature. While it is currently possible to design automobiles that are inherently much safer than the designs available today, such designs would not be affordable by the average driver. This situation leads to a search in areas other than automobile design for improved safety.

One such possibility is to avoid collisions in the first place, as opposed to protecting the occupants once a collision has occurred. Improvements in highway design, road markings and road signage can be aimed at this goal. Achieving such improvements inevitably requires a deep understanding of the cause of automobile collisions and a method for varying the circumstances of collisions to see how such changes may improve collision avoidance. Since crashing real cars into stationary objects, and into each other, is a dangerous, time consuming and expensive endeavor, computer simulation seems naturally suited to the task.

Another area that may benefit from computer simulation of automobile collisions is accident litigation where the ability of a jury to visualize a collision may be critical to the jury's verdict. Computer simulation is currently finding similar application in aircraft accident investigation and litigation. It is specifically the area of automobile collision visualization that the following research is intended to address.

The rest of the report is organized as follows. Section 3 provides an overview of purpose of the research and important contributions. Sections 4 and 5 discuss previous work and their limitations, respectively. In section 6, we present our new approach and discuss its superiority on other approaches. In section 7, we describe the implementation of our model. In this section, we discuss time versus event based and real time versus non-real time simulations. We provide the reason for choosing non-real time based simulation for the implementation of our model. We present the software languages and the libraries utilized. We also describe the physics and the collision detection technique. We explain

the generation of the scenarios and the vehicle models. In sections 8 and 9, we provide conclusions and some recommendations that further improve our simulator that can be considered in future work.

Objectives:

- Develop a tool that would enhance current techniques in crash analysis
- Animate and visualize crash occurrence to improve our understanding of crash causes
- Identify safety problems
- Propose solutions
- Test countermeasures

3. OVERVIEW OF RESEARCH OBJECTIVES AND CONTRIBUTION

Traffic crashes are major concern to the society. The cost and human suffering associated with traffic crashes are phenomenal. There is a need to improve our understanding of how and why traffic crashes occur, and hence improve our ability to better design and test countermeasures.

A computer simulation tool is developed within the framework of this project. Several elements of this project have been developed, including: visualization and animation capabilities including different view capabilities, collision detection, database search and classification, and geometric characteristics of the site. Other elements of the program has been partially developed because of the time and funding constraints, such as accident reconstruction, however, the core of the program is established and full addition of accident reconstruction capabilities is possible in an extension to this work. We also propose adding a learning capability to the program based on artificial Intelligence techniques, so that the model can detect if certain pattern of crashes tend to occur at certain location, and therefore identifying the problems with other similar locations. This would enable any designer to incorporate safety countermeasures in the design process without actually waiting and experiencing actual crashes and then develop countermeasures. Rather the model will be able to learn from existing locations' safety and derive what we can expect in other locations (what we refer to here as the "opposite to accident reconstruction"). In this research we have focused on intersections, because

more than half of the crashes occur at intersections or at the approach to intersections. It would be simple to extend the model to include other locations (e.g., freeway ramps, etc).

In this research project, we designed and implemented an automotive collision simulator that provides visualization and insight into collisions. Additionally, our simulator allows the rapid variation of accident conditions. The program has an intuitive graphical user interface that provides the user with full control over the simulation, by modifying the vehicle specification, path or the crash scenario (position, direction and speed of each object involved in the crash). It also allows for a variety of rendering options (e.g. texture mapping, wire-frame or solid polygonal models, light, camera position...etc.). The program has a modular scalable design and can run on different platforms. For collision detection, we used a collision detection tool called RAPID. We extended the functionality of RAPID to provide more accurate results. The program is a time-based simulator that uses actual vehicle models. We considered several specifications of the physical properties of the vehicles e.g. mass, rigidity, elasticity, angular moment and friction coefficient.

The program is easy to use and could be exploited in a variety of applications. The most important application is to use the program to improve the safety of automobiles, highways, and driver behavior. Another area could be using this tool as expert witness information in court to help with the presentation of an argument in a case.

4. PREVIOUS WORK

One of the first programs to simulate automobile collision was SMAC – Simulation Model of Automobile Collisions. This was done as a feasibility study at Cornell University. The researchers at Cornell were interested in demonstrating the feasibility of a mathematical model of automobile collisions, which could achieve improved uniformity and accuracy in the interpretation of evidence in automobile accidents. SMAC applications would give more accurate indications of collision severity. With SMAC the user had to estimate an impact speed to put into the model. This had to be input along with a number of other parameters that were not available. Because this information was needed a preprocessor was developed to provide the initial guess at impact speed and other parameters. This program was known as CRASH - the **R**econstruction of **A**ccident **S**peeds on the **H**ighway. This introduced the two most common methods of accident reconstruction techniques: damaged based reconstruction techniques and trajectory based reconstruction techniques.

The damaged based reconstruction technique uses information about the amount of deformation that occurred to the vehicle to reconstruct the crash. The information about the deformation is collected for multiple points and compared to the vehicle in its original form. The ability of the vehicle to resist crushing is then gathered from previous crash data. Then the crash impact speeds are calculated by deriving equations and using crash data.

The trajectory-based reconstruction technique is concerned with the separation of the two vehicles after the impact has occurred. Using the principal of Conservation of Momentum assumes that momentum preceding a collision and the system momentum after a collision, e.g. at separation, is conserved in the absence of external forces. Therefore, if we can determine the individual speeds and directions of motion that are required for each of the two partners in a collision to travel from separation to rest, then the direction and magnitude of this system momentum can be used to determine the magnitudes and directions of the velocities which must have existed prior to the collision, the impact velocities.

The SMAC and CRASH programs were developed under research contracts and are now maintained by McHenry Software ([3] and [4]). The National Center for Automobile Collisions has a model that uses these principles along with a Finite Element model of the automobile to crash test the car. Northwestern University is also conducting research in the area of automobile collisions.

5. LIMITATIONS OF CURRENT APPROACHES

The majority of the programs we encountered during our research were concerned with vehicle-to-vehicle collisions. These models required very complex amounts of input data in order to reconstruct the accident as accurately as possible. This is not ideal for a user that is interested in an investigation of many different possible scenarios for the resulting automobile collision. Input of data is cumbersome and there is only capability of examining a vehicle-to-vehicle collision. There were some applications that simulated crashes into objects other than cars. There were no models that allowed for many vehicles in the model. Also many of the programs produced many numbers that did not directly output visual information. Many models have after the fact rendering and no control over the viewpoint of the collision.

6. THE NEW APPROACH: THE CRASH SIMULATOR

In our approach, we designed and implemented the Crash simulator that have the following features:

6.1. Intuitive Graphical User Interface (GUI):

Which allowed the user to easily modify the input parameters, through a number of windows. This includes the vehicle specification and the scenario of the collision. The user friendly GUI makes the data input phase much easier. It allows accommodating different types of users, even those who do not have any computer experience. Figure 1 shows The program's main user interface. It allows the user to specify the number of vehicles involved, camera position, rendering options, and the animation time slice. The rendering options allow the user to choose wire frame or solid rendering, smooth or flat shading, and light or texture mapping.

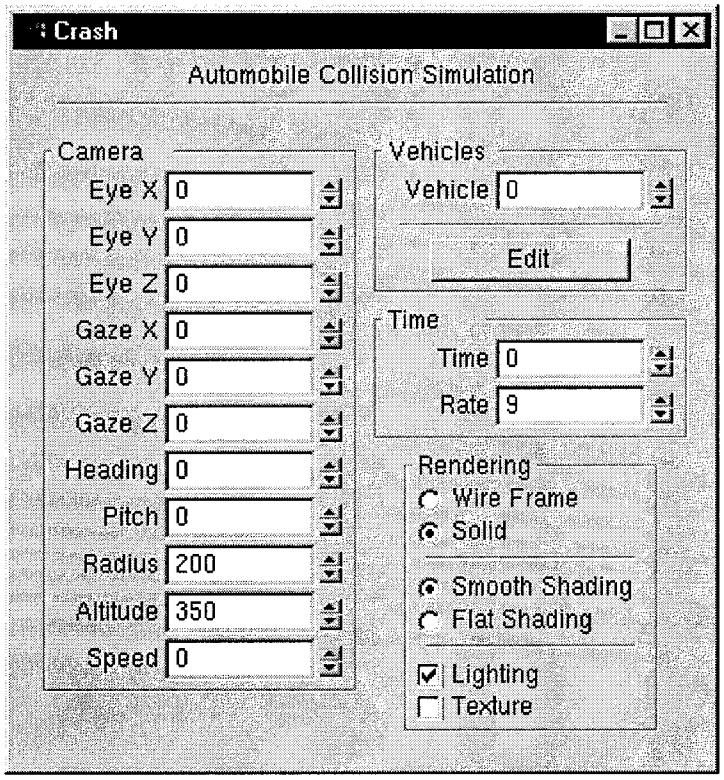


Figure 1: Crash Control

There are several others windows that will be explained in section 6.2.

6.2.Simulation Control:

Our simulator gives the user full control over the simulation process. The user can directly watch the consequences of changing any of the input parameters. The user can do that through several windows: Intersection geometry control, car control, view control and action control.

6.2.1 Intersection Control:

Figure 2 shows the intersection geometry control. The Intersection Geometry module allows users to change intersection configuration. Figure 2 shows an example of a 4-leg

intersection with major and minor roads. Figure 3 shows the view of the intersection specified in Figure 2. The module have some default values for the lane width, turning lane's length, taper length, median width, and pavement marking. The initial configuration of the intersection is stored in an ASCII file called *road.def.* The user is allowed to change the default configuration before or during simulation. For example, if the user wanted to study the effect of changing the geometry of the intersection on the accident, then he/she is allowed to change the configuration even during simulation. Our simulator will then change the rendering of the intersection geometry to reflect the new configuration.

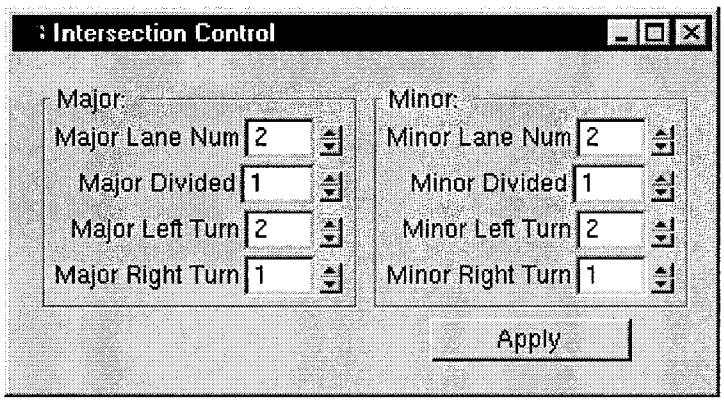


Figure 2: Intersection Geometry Control

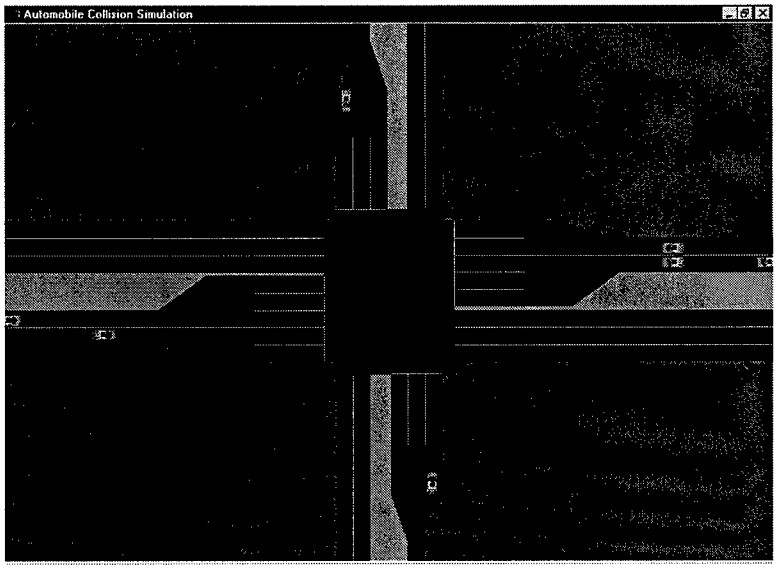


Figure 3: Intersection Plan (Orthogonal View)

6.2.2 Car Control:

The Car Control module allows users to set the initial speed, direction, and vehicle position for each vehicle involved as shown in Figure 4. The vehicle speeds are in kilometeres per hour (kph). The intial vehicle coordinates (Xpos and Ypos) are measured from the intersection point of the center lines of the roads. The direction field defines the movement direction of the vehcile. For example, a zero direction means that vehicle direction is from left to right (east direction). Information of each vehicle is stored in an ASCII file. The user is also allowed to change the initial configuration before or during simulation.

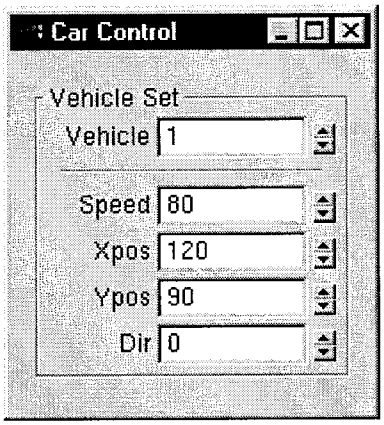


Figure 4: Car Control Module

6.2.3 View Control Module

The View Control Module shown in Figure 5 allows users to see the intersection in different views. It also allows use for tracking a specific vehicle (Figure 6). Figure 3 shows an orthographic view. In addition to these two options, there are several other options as shown in Figure 5. The Drive option allows users to visualize accidents from the driver's view as shown in Figure 7. The Orbit view is like the orthographic view but it keeps rotating the scene to allow the user to view it from several angles as shown in Figure 8. The Fly view shown in Figure 9 is also an orthographic view but it tracks ("flies with") the vehicles. It does not only view the center of the intersection as in the orthographic view. In Fly view not only the vehicles are moving but the camera is also moving along. To illustrate the difference between the orthographic view and the Fly view, we took two screen captures of the Fly view, at two points of time. This is shown in Figures 9 and 10 that illustrates two points x and y of time.

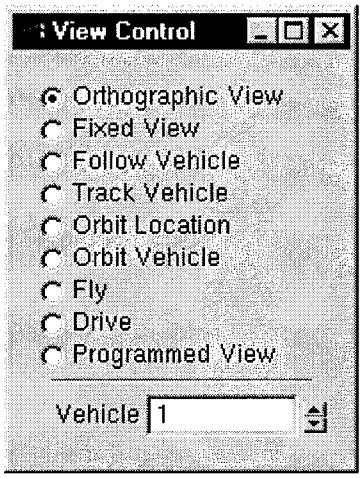


Figure 5: View Control Module

Figure 6: Track-Vehicle View

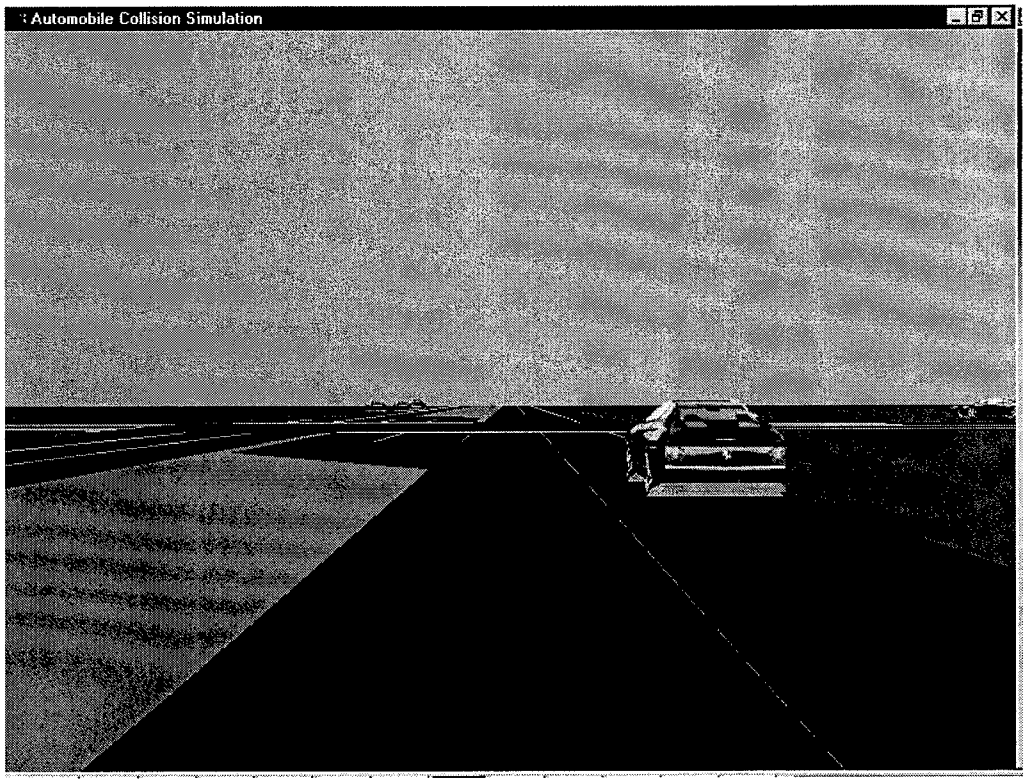


Figure 7: Drive View

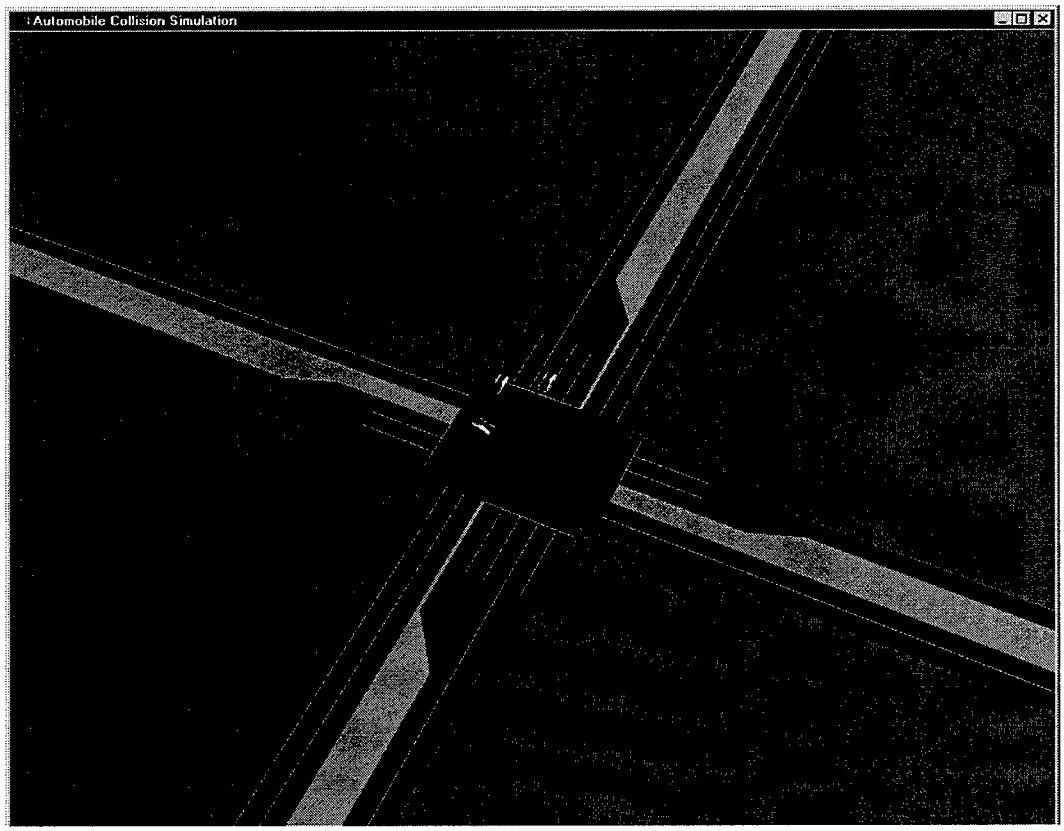


Figure 8: Orbit view

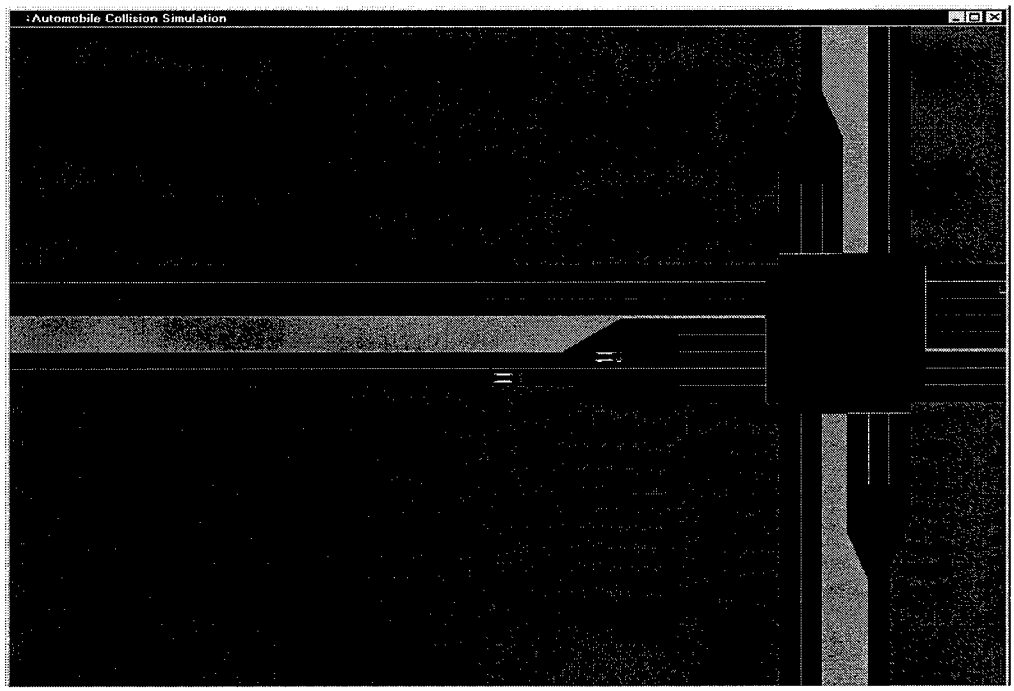


Figure 9: Fly View at point x in time

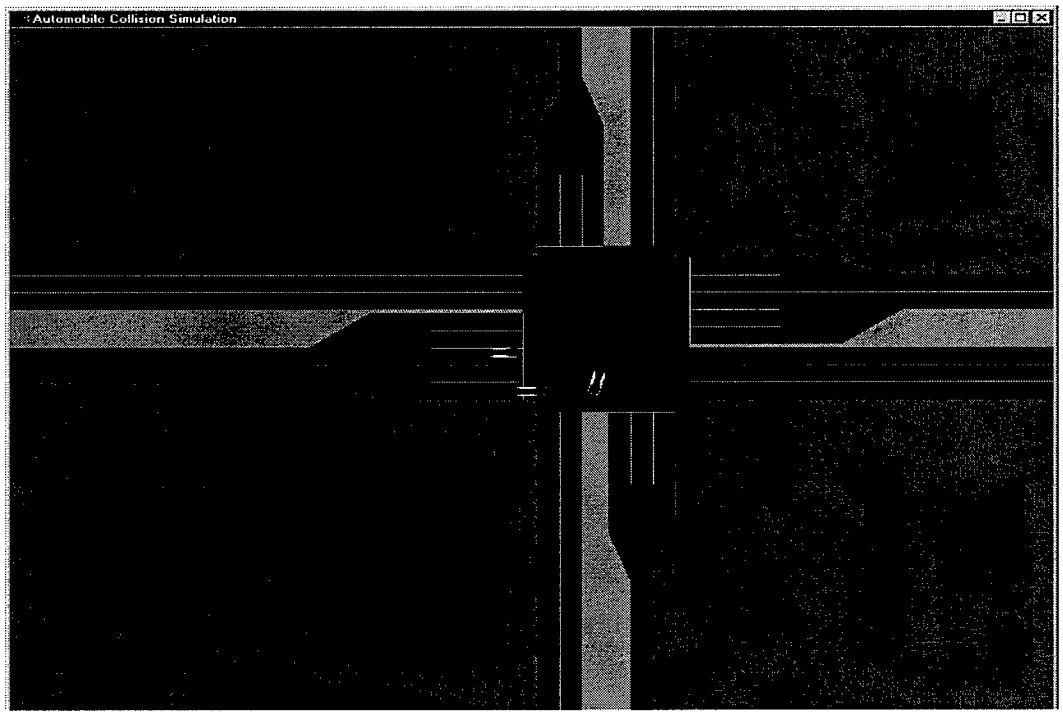


Figure 10: Fly View at point y in time

6.2.4 Action Control Module

The Action Control Module provides the user with a simple window to control the crash simulation as shown in Figure 11. The Stop button allows user to stop the simulation run at any time during simulation. The Go button starts the simulation. The Step button runs the crash in step intervals. The Reset button is used to restart the simulation and the Quit button is used to exit the simulator.

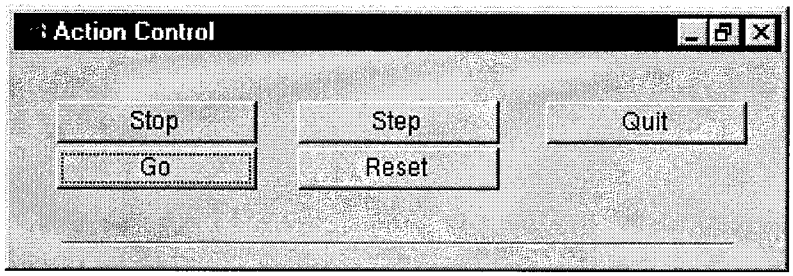
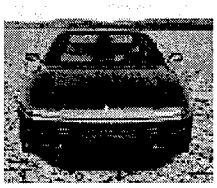


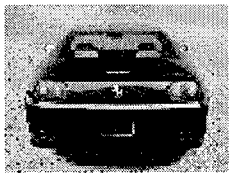
Figure 11: Run Control Module

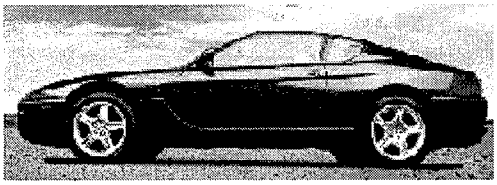
6.3 Crash Visualization: our approach provides a realistic visualization of vehicle collision. This provides an excellent opportunity to the user to know what exactly happened in a crash. It might help the user reconstruct the precise circumstances of the accident, or investigate how the outcome of an accident may have been different under different circumstances. Figure 12 illustrates a visualization of a collision between two vehicles.

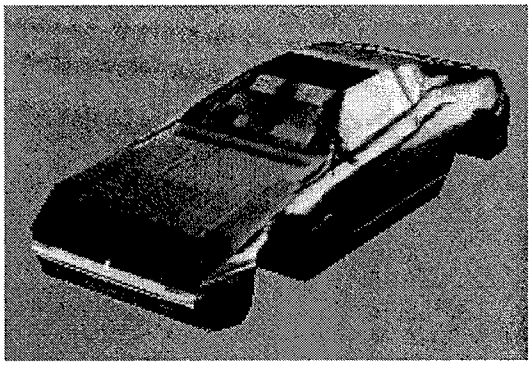
Figure 12: Collision Visualization

6.4 Vehicle Models:









We use car models (not just any objects as used in other approaches) to better envision crashes.

6.5 Modularity:

Our simulator has a modular design that allows for the extensibility of any of the features it currently provides. This feature results from the choice of using an object-oriented programming language, which is C++. As an object programming language, C++ has the advantages that it models real-world objects as software objects. For example, a vehicle is an object, and an intersection is another object. This will result in many benefits. One is that it allows the program to be easily extended to provide more features. Two, is it

facilitates the development specially if many programmers are involved. Three is that it makes the program manageable because it is divided into smaller modules.

6.6 Multi-platform Support:

Our simulator can be used on many platforms as illustrated later. It can be used on a PC, Macintosh, SGI, ...etc. That is it can run on several operating systems UNIX, Linux, Irix, Windows, Windows NT or MacOS. This gives our simulator its portability feature.

6.7 Arbitrary Number of Vehicles:

Since accidents can be very different in nature, the decision was made in an early stage that the user should have the option to simulate a collision with an arbitrary number of vehicles involved. Apart from the vehicles involved in the collision, the user should also have the option of including other objects in the scene and or in the collision. The objects can include anything from trees or lampposts to houses or people. This gives our design a very important feature: scalability.

6.8 Vehicle Path Control:

The vehicles in the collision simulations should be controlled in one of two ways, either given a predefined route through a scenario file passed to the program or manually through the user interface provided by the application.

In both, the scenario file and the user interface, the user can choose to let the car change position, direction or speed.

6.9 Different Visualization Features:

The user chooses how the collision scene should be rendered through the user interface, and the user can change the rendering method at any point during the execution of the program. This was discussed in sections 6.1 and 6.2.

The different rendering methods include texture mapping, wire-frame and solid polygonal models. When using the solid model rendering, the user should be able to include lights in the scene and control the position and intensity of the light(s) from the user interface.

The user should also be able to choose different camera angles or let the camera track one of the vehicles during the simulation of the collision.

The application also includes features to step through a scenario and enable playback of parts or the whole scenario for more accurate analysis of why and how the collision occurred.

6.10. The Database Query Mode:

Our simulator has two modes of execution. One is the animation mode and the other is a database query mode. In the animation mode, the user can view an animation of a scenario that is already known to him/her. That is the user can use either the default parameters or change them using the graphical user interface. The user in this case can also use the accident report number. In the query mode the user does not know all specifics of a particular accident, but knows some of the circumstances of the accident. For example, the number of vehicles involved, accident type, location, date, weather

condition, lighting conditions, intersection specifications ...etc. The user then can run the program in the query mode and enter some information about the accident he/she is interested in animating. Figure 13 shows the database query interface. Our simulator will then search the accident scenario database and will match them with the user's information. The user is allowed to choose from the matching scenarios which accident he/she wants to visualize. The simulator will animate the user's selected accident.

Using the report number, we can access the database file. The accident database includes accident-related data such as date, time, accident type, location. The program will make some queries. e.g., one can make a query to animate (1) daytime accidents, (2) accidents in a certain weather condition, and/or (3) accidents that involve a specific number of vehicles.

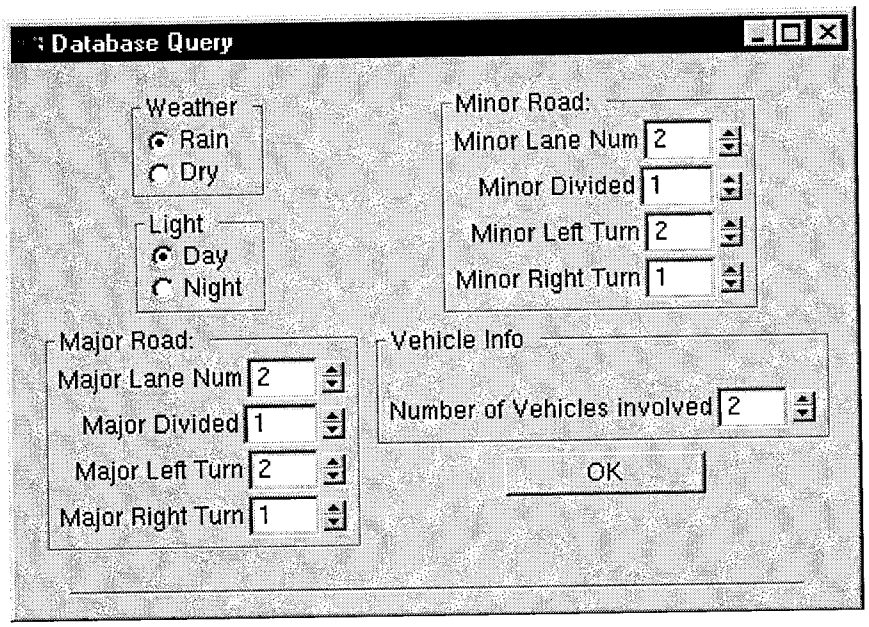


Figure 13: Database Query Interface

7. IMPLEMENTATION

7.1 Time Based Simulation

A time based, non-real time, modeling approach has been taken in the implementation of the collision simulation. A time based approach was chosen, as opposed to an event based approach, in order to accurately represent the physics of a collision including vehicle body and chassis deformation. A non-real time simulation approach was chosen in order to support computer equipment that is commonly available. A review of the computations involved in the collision physics, and the graphics requirements for realistic three-dimensional visualization, indicated that high end computational and image generation equipment would be required for real time operation. Additionally, real time operation is not required since the user is not attempting to control vehicle movements directly in real time. This also has the advantage that our simulator does not need high-end hardware (the program works on a PC, workstation, or an SGI machine).

7.2. Software Language & Libraries

The desire was to target as many platforms and operating systems as possible, both to increase the utility of the software and to ease development constraints. Additionally it was desired to use object-oriented methodologies for software implementation in order to give our simulator its modular and extensible features. To support these requirements C++ was chosen as an object-oriented language available on all development platforms.

OpenGL was chosen as the graphics library due to the advanced rendering capabilities it offers and to its availability on all development platforms. The GL Utility Toolkit (GLUT) was used to maintain windowing system independence not possible with X, Microsoft Windows, etc. The user interface was constructed using the GLUT User Interface library (GLUI) which is implemented entirely using GLUT thus maintaining windowing system independence while providing more advanced user interface features than are available in GLUT alone.

7.3. Physics

The Crash application allows for several specifications of the physical properties of an object. The following factors are input for each object:

- Mass
- Angular moment about each model coordinate axis
- Rigidity
- Elasticity
- Friction coefficient

Rigidity is a measure of the force, in Newtons, required to deform any vertex of a given object one meter in any direction. This is similar to a spring constant, except that there is no assumption that the vertex will seek to return to its original location. **Elasticity** is a measure of the latter, the tendency of a vertex to rebound after a deformation, specified by a value from zero to one. An elasticity of zero indicates that a vertex will remain

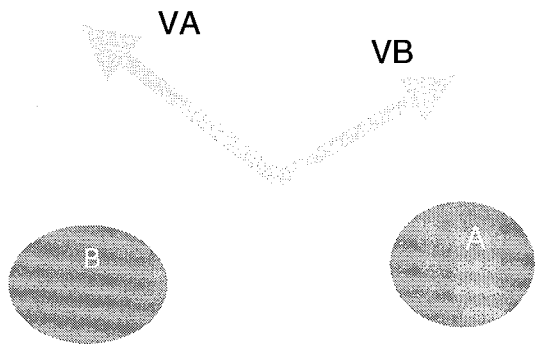
exactly in the position to which it is deformed (complete distortion), while an elasticity of one indicates that it will completely regain its original location (no distortion). The vertices of an object were modeled as the endpoints in a system of damped springs.

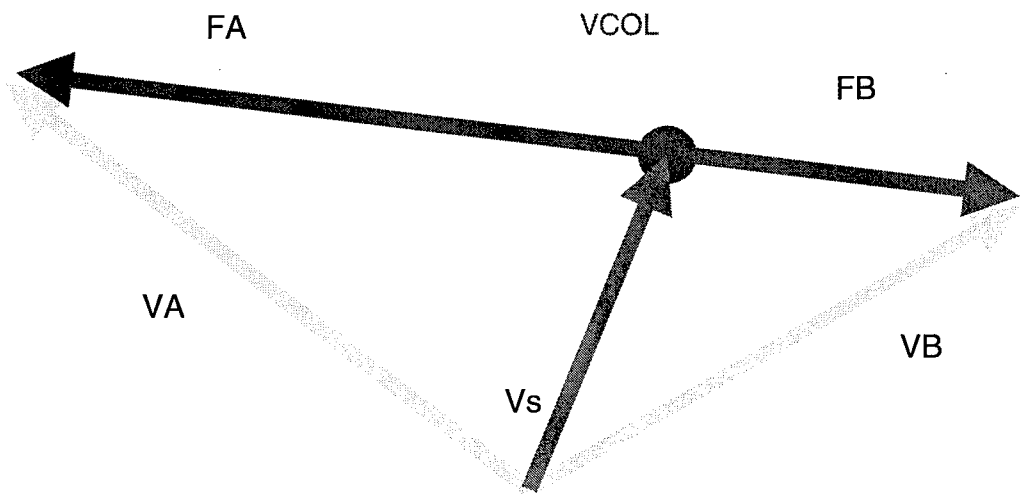
The assumption is made that any elastic recovery occurs during the same time slice as the deformation, and is therefore factored into that deformation. This assumption is safe, in the sense that the simulated objects (cars) are relatively stiff, and the small amount of elastic recovery that occurs is brief, relative to a time slice. Furthermore, the elastic recovery (which causes two objects to continue to overlap momentarily) is not visible until the colliding objects actually separate. It is the continued detection of contact after two objects cease to move toward one another that causes the Crash application to continue to exert force on both objects. It is this continued exertion of force makes the objects move apart. In the absence of any elastic recovery by individual vertices when they are deformed, the overall collision between two objects would be non-elastic, and they would not rebound after impact.

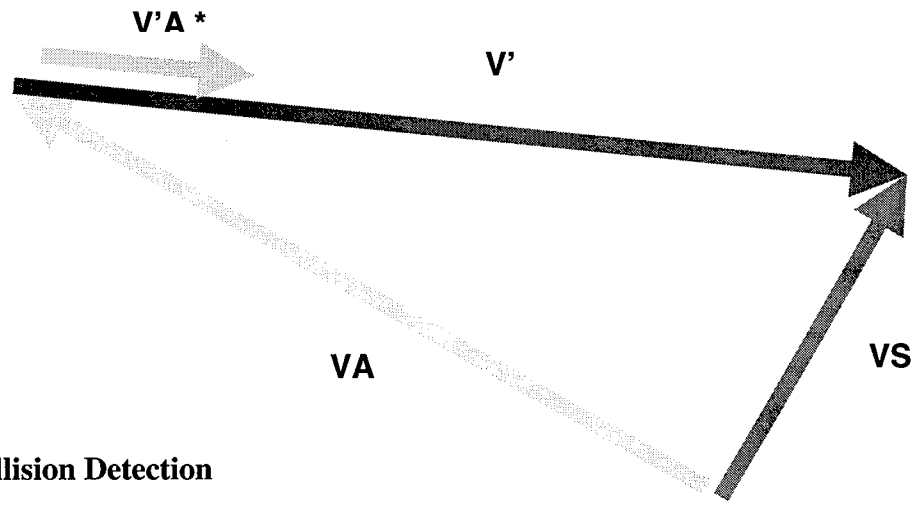
Along with mass, input to the application specifies the moment of inertia for rotation of an object about each of the three canonical model coordinate axes. The specific moment for rotation about an arbitrary axis passing through the origin is approximated by taking a vector in the direction of the axis of rotation, and projecting onto it each of the canonical moments. The specific moment is taken to be the sum root mean squares of the magnitudes of these projections. This is equivalent to considering the object to be an ellipsoid. While this is generally invalid, the error is typically of the same order of

magnitude as what would be introduced if the shape of the object were actually analyzed, but the object was still assumed to be of uniform density.

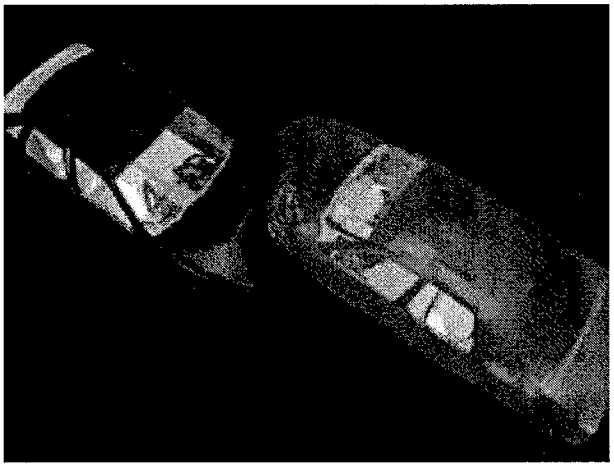
The following illustrates the non-elastic collision of non-deformable objects:







7.4.Collision Detection



For collision detection, we used the Robust, Accurate Polygon Interference Detection (RAPID) [1] collision detection tool developed at the University of North Carolina. For each modeled object, RAPID creates a data structure from the model coordinates of a list of triangles, each of which is designated by a unique integer index provided to RAPID

along with the coordinates. The interface to RAPID provides an operation to which a client program specifies two such object models, along with a point and a 3x3 transformation matrix which specify each object's world coordinate position and orientation. The operation merely generates a list of pairs of triangles, one from each object that are in intersection with one another. There is no assumption that the triangles specified in the object models form a continuous surface, or even that they form a surface at all.

RAPID internally creates a hierarchy of oriented bounding boxes which define the extent of modeled objects. It uses these to perform initial rough comparisons between two objects, to determine whether it is necessary to perform the expensive additional processing which compares individual triangle pairs in the objects. Although RAPID is not unique in its use of oriented bounding boxes, other collision detection algorithms exist [2] which use bounding boxes which are aligned to the model coordinate axes. Clearly the use of oriented bounding boxes allows for a tighter fit around an arbitrarily shaped object using smaller boxes. The result is improved due to fewer unneeded comparisons.

The input to and output from RAPID were kept as simple as possible, with the expectation that the client application would perform any processing needed as a result of a detected collision. As few assumptions as possible were made as to what that processing might entail. An application that required a high degree of accuracy, for example, would generally take a pair of triangles reported by RAPID to have collided,

and compare them to determine the actual intersection. With that known, the simulation could be reversed to an (interpolated) exact moment of impact, and an exact point of impact could thus be derived.

The authors of RAPID have also developed higher level tools - ICOLLIDE and VCOLLIDE - which use RAPID to perform basic collision detection. These tools perform additional processing, such as avoiding unneeded comparisons and managing a large number of objects, but also place restrictions on their use, such as the assumption that the intersecting triangles do in fact form a convex polyhedron.

The Crash simulator performs some approximations (or assumptions) in order to utilize the basic responses from RAPID:

1. The collision between two triangles, and the time slice in which it occurs, are treated atomically.
2. When two objects collide, and RAPID reports collisions between pairs of individual triangles, the assumption is made that each triangle on one of the objects collided with "the other object" rather than with "the other triangle". Thus the processing of that triangle is performed based on the properties of the other object as a whole, or on the collision as a whole, rather than on the particulars of the other triangle with which it actually intersected.

3. Since RAPID does not directly compute the actual world-coordinate location of the triangles, we assume that a reported triangle have made contact with the other object one time slice ago.

It is very important to note here that to enhance the output from RAPID, considering that RAPID does not include occluded triangles – those that are completely inside the other object - in its output. The collision detection of our Crash application discovers such triangles by performing a "flood fill" algorithm to identify triangles that are contained within a ring of triangles already reported to have collided. For such triangles, there is no pairing with a triangle from the other object.

Two objects will typically collide and deform one another over a period of several time slices. Collision detection is performed at each time slice, and for all but the first, the objects are already in contact. Thus the vast majority of detected collisions are between objects that are in contact throughout the entire time slice. This makes assumption 3 a closer approximation than would be the assumption that an initial collision between two objects occurs one-half time slice prior to its detection (which might appear to be statistically more correct).

A vertex of that triangle is deformed by a directed distance that is a function of the motion of both objects during that time slice, and of the physical properties of both objects. Once a vertex is deformed, the impulse required to have caused the deformation in the given amount of time is calculated. It is this impulse that, along with the impulses

from all other deformed vertices, is applied to the overall object to re-compute its linear and angular velocity for the next time slice.

7.5. Scenario Generation

An important part of our simulation was to reproduce accident scenarios. We had a scenario file for each object involved in the crash. This guarantees the scalability of our model in the sense that it is not limited to a certain number or type of objects. Objects could be vehicles, telephone poles, lampposts, trees, ...etc. For vehicles, there are two types of scenario generation files. The first constitutes two parts:

1. The initial condition of the vehicle
2. The event description.

The initial condition includes the initial velocity in km/hr. The initial position of the vehicle <x, y, z> where x, y, z are normalized to be ε [-1, 1], the initial value of z=0. The direction of the vehicle in degrees which is equal to the angle off the North in clockwise direction. The event description included the value and the time the event occurred. There are three possibilities for an event: a driver can accelerate, brake or turn the steering wheel. In our model these events correspond to Gas, Brake and Steering wheel. The value of the Gas means the increment of acceleration from the last velocity value. It is normalized to have values between 0 and 1, where 0 means no acceleration and 1 means maximum acceleration. The value of the Brake event means that the driver applied the brakes at this time. It also can have a value between 0 and 1, where 0 means no brake was

applied and 1 means full brake was applied. The Steering Wheel event is recorded if the driver turned the steering wheel right or left. It is given values between -1 and +1, where -1 means 90 degrees to the left and +1 means 90 degrees to the right

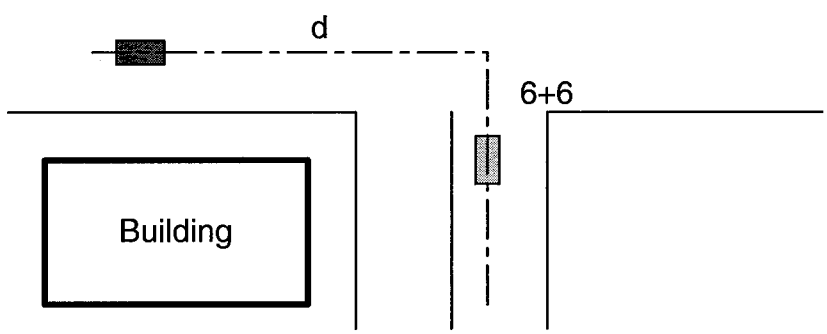
.

In the second type, the scenario file describes a predefined route for every vehicle. That is it specifies at regular intervals of time each vehicle speed, position and direction. It is very important to note here that we allow the user to alter the scenario files through the user interface. Thus these files are not only readable but are also writeable. This provides the user with more flexibility and achieves better results in terms of usability that complies with our goals for that design.

<u>Example for a crash scenario at an intersection</u>:

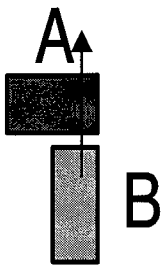
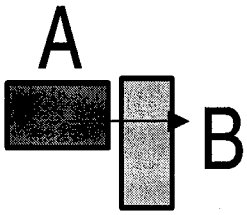
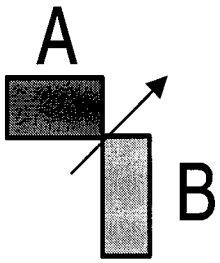
Cases of Angle collisions

Case 1 : Insufficient Amber Time

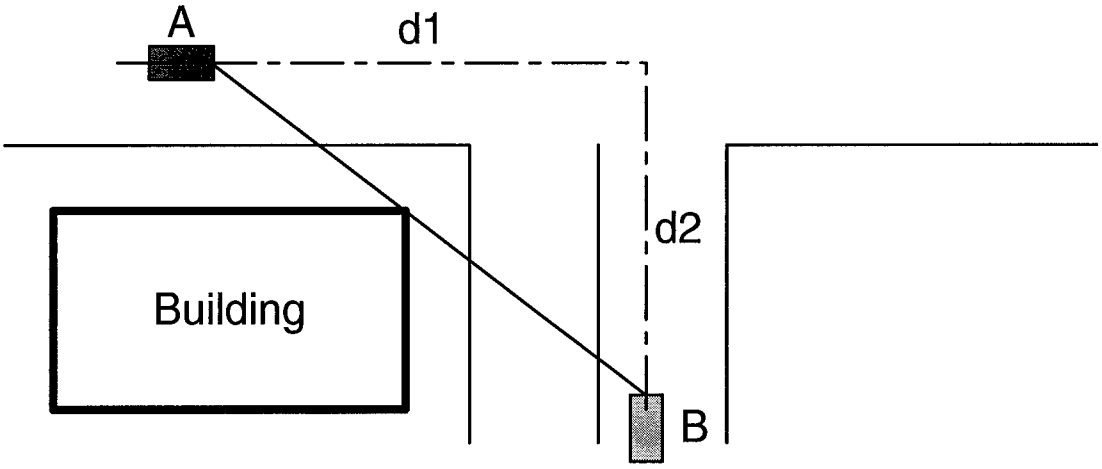


- Car A travels with the approach speed during the yellow phase or at the initiation of the red phase.
- Car B starts moving from stop when light turns green and accelerate with a rate depending on the horsepower.
- Stop line is at a distance of 6' from the edge of the crossing road

Analysis of the manner of collision

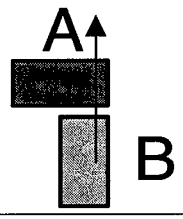
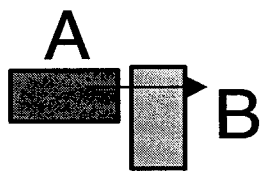
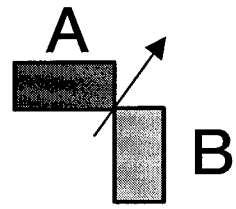
		
Type I	**Type II**	**Type III**
B hits A	**A hits B**	**A & B hit each other**

Case 2 : Insufficient Sight Distance



- Both car A and B travel with the approach speed.
- Both car A and B will try to stop once they detect that no sufficient distance

Analysis of the manner of collision

		
Type I	**Type II**	**Type III**
B hits A	**A hits B**	**A & B hit each other**

7.6.Vehicle Models

To put vehicle data into the model we set up a text file that contained the pertinent data relative to the crash. The vehicle file allows the user to input surface point data into the program. The file will contain one car object that has surface point data, polygon data, texture map information, and physics information. The car object consists of a list of vertices that are referenced by the polygons. The polygons are input as polygon lists. Each polygon list is a part of the car body. A texture map can be assigned to each of those polygon lists. This allows the user to input polygon lists for the front, rear, top, bottom and sides of the vehicle and put a texture map on each polygon list. The physics data is data that includes the center of gravity, mass, stiffness, etc. so the car can be properly deformed when a collision occurs. This allows the customization of the car models so they will not only act like real cars but they will also look like real cars.

8. SUGGESTIONS FOR THE IMPROVEMENT OF THE COMPUTER PROGRAM

In this report a computer crash simulation tool was presented. This tool is an animation tool that provides visualization of automotive collision. Its ease of use and the plenty of features that it provides make it very useful for a variety of applications. In the rest of this section, we will present some recommendations that can help to improve our simulator in future work. We will also discuss some of the problems we faced.

This model can be improved in the future with the addition of more detailed vehicle models. These models could include internal parts, each with their own physical properties.

The achievement of accuracy can be further delegated to the simulation input. That is, if the simulation does not portray real events with sufficient accuracy, improvement is possible by refining the input. For example, one could

- Subdivide the triangles in the model, or make them more uniform in size,
- Describe the modeled object hierarchically, with distinct physical properties specified for each component,
- Run a simulation using smaller time slices.

Modeling the vertices of an object as the endpoints in a system of damped springs can have the disadvantage that even heavily damped springs return completely to equilibrium.

Although such a model would correctly simulate a partially elastic collision, any deformations would ultimately disappear. Critical to this simulation was the irreversible distortion of the model. However, if the objects were given no elasticity, they would collide like two wet towels, which is equally unrealistic. A refined Crash application may implement a damped spring model, but must continue to allow permanent, irreversible distortion of those springs.

One of the major difficulties that we faced during our implementation of this tool involved the accumulation of the impulses from the individual vertices and the conversion of the net impulse into linear and angular acceleration. A force vector associated with a particular vertex can be separated into two components, one directed toward the center of gravity, and the other orthogonal to it. It was at first assumed that the former would translate into linear acceleration while the second would result in angular acceleration. This is demonstrably false. Consider a steel rod at rest. An impulse to the end of the rod, exactly perpendicular to the rod causes the rod to spin, but the center of gravity of the rod does not remain motionless. It is propelled in the direction of the impulse. Thus a portion of the impulse is translated into linear motion, even when the force is orthogonal to the center of gravity. The calculus provides the correct resolution of this impulse into both angular and linear motion. We developed a reasonable discrete approximation. The approximation is based on a comparison between the instantaneous forces required to move a vertex an arbitrarily small amount, assuming first that the entire object is moved in a linear manner, and then that the entire object is rotated about its center of gravity. Suppose, for example, that it would take 9 Newtons to accelerate an

object by 1 meter per second squared, and at a particular point on the surface of the object it would take 1 Newton to cause a rotational acceleration that would result in the same local instantaneous linear acceleration. Then a force applied to the point in a direction that is orthogonal to the direction of the center of gravity from that point is resolved to 10% linear acceleration and 90% angular acceleration.

For the problem of combining the angular impulses from all the vertices of an object and effecting a single change to the linear and angular velocity of the object, it must be expected that the vertices to which forces are applied in a given time slice are not local to a particular region of an object. Since a car object will typically have four tires in contact with the road, there are four regions of contact even when no other cars are nearby. Consider again the steel rod. Suppose it is resting so that only its two ends are in contact with two supports. Then an identical upward force is applied to each end of the rod, perpendicular to the center of gravity. Together they exactly offset the force of gravity on the rod. Each upward force by itself would result in mostly angular motion, but the net result of the two forces is zero angular force, and a linear force equal to their sum. This illustrates the need for an operation that can sum all of the forces on an object in a manner that preserves the ability to correctly derive the total linear and angular acceleration. A good approach would compare a pair of force vectors, first extract and combine the component of each that is known to cause linear motion. That component should be subtracted from the original vectors and stored separately. The remains of the original forces are then compared to determine a component that would be opposing angular force. That component should be evaluated for possible conversion to linear

motion, subtracted and stored with the previously extracted linear force. The remains of the original force vectors are then independent angular forces. However, they are only independent of one another. They must similarly be made independent of all other force vectors acting on the object.

9. CONCLUSIONS AND RECOMMENDATIONS

A computer simulation tool is developed. Several elements of this project have been developed, including: visualization and animation capabilities including different view capabilities, collision detection, database search and classification, and geometric characteristics of the site. Other elements of the program has been partially developed because of the time and funding constraints, such as accident reconstruction, however, the core of the program is established and full addition of accident reconstruction capabilities is possible in an extension to this work. We also propose adding a learning capability to the program based on artificial Intelligence techniques, so that the model can detect if certain pattern of crashes tend to occur at certain location, and therefore identifying the problems with other similar locations. This would enable any designer to incorporate safety countermeasures in the design process without actually waiting and experiencing actual crashes and then develop countermeasures. Rather the model will be able to learn from existing locations' safety and derive what we can expect in other locations. This would be the most important extension to this work: the ability of the computer model to learn from crashes that are animated at the different types of intersections, have the ability to generalize and associate specific types of crashes with specific design, location or circumstances. With this addition any designer will be able to detect safety problems at any location easily and efficiently.

In this research we have focused on intersections, because more than half of the crashes occur at intersections or at the approach to intersections. It would be simple to extend the model to include other locations (e.g., freeway ramps, etc).

10. References

[1] University of North Carolina, Computer Science Dept.

http://www.cs.unc.edu/~dm/collide.html

[2] Lin, Ming C. Efficient Collision Detection for Animation and Robotics, University of California, Berkeley, CA.dissertation 1993.

[3] McHenry Software Homepage, McHenry Software, Inc.

http://www.mchenrysoftware.com

[4] CRASH – 97-Refinement of the Collision Algorithm, McHenry, B.G., McHenry, R.R., SAE paper no. 97-0960, 1997 SAE Congress.