AUTOMATED VEHICLE ARTICULATION AND ANIMATION:

A MAXSCRIPT APPROACH

A Thesis

by

CHRISTOPHER COREY GRIFFIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Visualization Sciences

AUTOMATED VEHICLE ARTICULATION AND ANIMATION:

A MAXSCRIPT APPROACH

A Thesis

by

CHRISTOPHER GRIFFIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Tim McLaughlin
Committee Members,	Frederic I. Parke
	John Keyser
Head of Department,	Tim McLaughlin

December 2010

Major Subject: Visualization Sciences

ABSTRACT

Automated Vehicle Articulation and Animation:

A Maxscript Approach. (December 2010)

Christopher Corey Griffin, B.E.D., Texas A&M University

Chair of Advisory Committee: Tim McLaughlin

This thesis presents an efficient, animation production-centric solution to the articulation and animation of computer generated automobiles for creating animations with a high degree of believability. The thesis has two main foci which include an automated and customizable articulation system for automobile models and a vehicle animation system that utilizes minimal simulation techniques. The primary contribution of this thesis is the definition of a computer graphics animation software program that utilizes simulation and key-frame methods for defining vehicle motion. There is an emphasis on maintaining efficiency to prevent long wait times during the animation process and allow for immediate interactivity. The program, when implemented, allows for animation of a vehicle with minimal input and setup. These automated tools could make animating an automobile, or multiple automobiles of varying form and dimensions much more efficient and believable in a film, animation, or game production environment.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION..... 1

II RELATED WORK..... 4

II.1. Film and Animation..... 4
II.1.1. MadCar..... 4
II.1.2. EzCarRig..... 5
II.1.3. Smart Cars..... 6
II.2. Automated Articulation..... 7
II.3. Autonomous Vehicles..... 8
II.3.1. Virtual Racing..... 8
II.3.2. Autonomous Real-world Vehicles..... 9
II.4. Virtual Automobile Dynamics..... 10
II.5. Real-world Automobile Dynamics..... 11
II.5.1. Darwinian / Ackermann Steering Geometry..... 11
II.5.2. Maneuvering Vehicles..... 12

III METHODOLOGY..... 14

III.1. Automobile Articulation..... 14
III.1.1. Rig Template..... 14
III.1.2. Fitting Articulation to Geometry..... 15
III.2. Automated Automobile Animation..... 16
III.2.1. Driving Controls..... 16
III.2.2. Path Step..... 17
III.2.3. Bezier Curve..... 18
III.2.4. Turning Radius..... 21
III.2.5. Osculating Circle..... 22
III.3. Automated Ground Detection..... 23
III.4. Secondary Animation..... 24
III.5. Darwinian / Ackermann Steering Geometry..... 25

IV IMPLEMENTATION..... 27

IV.1. Implementation Environment..... 27
IV.2. Automated Articulation..... 28
IV.2.1. Initialization..... 28
IV.2.2. Rig Building..... 28
IV.2.3. Geometry Fitting..... 29
IV.2.4. Save / Load Rig..... 32

CHAPTER Page

IV.3. Automated Animation..... 33
IV.3.1. Callback Functions..... 33
IV.3.2. Bezier Curve Functions..... 34
IV.3.3. Simulation Curve Functions..... 35
IV.4. Automated Ground Detection..... 38
IV.4.1. Zero Plane Projection..... 38
IV.4.2. Raycasting..... 38
IV.5. Automated Automobile Dynamics..... 40
IV.5.1. Secondary Animation..... 40
IV.5.2. Ackermann Geometry..... 41

V RESULTS..... 43

V.1. Specifications..... 43
V.2. Animation Sequence..... 44
V.2.1. Articulation..... 44
V.2.2. Animation..... 45

VI FUTURE WORK AND CONCLUSION..... 47

VI.1. Future Work..... 47
VI.2. Conclusion..... 47

REFERENCES..... 49

VITA..... 51

LIST OF FIGURES

FIGURE		Page
1	The Darwinian Criterion	12
2	Path Step	18
3	Bezier Curve	19
4	Bezier Controls	20
5	Turning Radius	21
6	Osculating Circle	23
7	Secondary Animation	24
8	New Rig	28
9	GenGeo	29
10	Wheel Measurements	30
11	GenGeo Complete	31
12	Save Rig	32
13	Bezier Rig	34
14	Zero Rig	35
15	Redefining the Animation Path	36
16	Ground Detection	39
17	Animation Sequence Automobiles	44
18	Chase!	45

CHAPTER I

INTRODUCTION

Animated computer-generated automobiles are prevalent in today's entertainment industries. Examples can be seen in the digitally recreated car races in the feature film *Speed Racer*, the urban demolition derby of the *Grand Theft Auto* video game series or the living, breathing automotive characters of the feature animated film *Cars*.

When it comes to the animation of a real-world object such as an automobile, the believability of motion contributes significantly to the viewer’s experience. Just as in the case of human animation, vehicle animation is held to a higher standard of criticism due to people's everyday interaction with automobiles. Meeting these high standards is a difficult task that is achieved by accounting for the physical attributes and limitations of automobiles such as velocity, turning radius, pitch, and roll. It is a time consuming and tedious effort to process all these physical variables manually, and can be overwhelming for an animator in a production environment.

Like all real-world objects, an automobile's motion is governed by physical limitations. If these limitations are not respected in the motion of the vehicle, the results may be unconvincing to an audience. These limitations are typically determined by the dimensions of the vehicle and the positioning of key mechanical components including the suspension arms and wheel knuckles. Hence, automobiles of varying dimensions have distinguishing limitations and therefore unique motion patterns.

This thesis follows the style of *IEEE Transactions on Visualization and Computer Graphics*.

Accounting for appropriate secondary motion of automobiles in a production environment is difficult as well. In vehicle dynamics, secondary motions such as pitch, roll and yaw rely heavily on physical calculations and the knowledge of physical variables unique to an automobile such as the mass of key components and the structure and spring coefficients of the suspension system. To manually calculate all the physical forces acting on an automobile at all instances while it is in motion would be impractical in a production.

To make a vehicle appear to drive accurately over complex terrain is another problem prevalent in automobile animation. Again, this is a case where large amounts of physical calculations would be required to get the desired results. Alternatively, to do this animation manually would be time-consuming and negate immediate and production-efficient editing changes.

The approach presented in this research alleviates several issues of automobile articulation and animation; including the problem of variable automobile model dimensions, limitation-based vehicle motion patterns and physically-based animation effects such as secondary motion and ground detection. These issues are addressed with the use of procedural and non-procedural techniques.

To answer the problem of varying automobile dimensions, an automated articulation system was developed that utilizes geometry fitting techniques and allows for rig dimension customization. When implemented, this system would allow for the immediate articulation and animation of common automobile models of nearly any dimension.

To resolve the issue of limitation-based vehicle motion, procedural functions based on the real-world limitations and predictable motion patterns of automobiles were implemented in conjunction with key-frame methods. These functions would allow for the recreation of more accurate automobile motion.

To address the complications associated with physically-based animation effects such as secondary motion and ground detection, complex physical calculations are simplified and combined with procedural functions based on basic geometric calculations. This method provides a production-friendly solution and believable animation effects without the computational overhead and production delays that commonly occur with conventional simulation techniques.

The techniques in this research differ in many ways from the methods already applied to automobile articulation and animation. A key difference is that several past approaches rely on a nearly exclusive implementation of one general technique, such as simulation, path-interpolation or key frame animation. This research uses a combination of all these techniques in orchestration to achieve the desired results.

This thesis focuses on the optimization of automobile articulation and animation. The central motivation is the perceived realism, or believability, of vehicle motion rather than the absolute physical accuracy of automobile animation. This ideal is achieved in many aspects through the exploitation of simplified geometric models that can easily be applied to automobile structure and motion. The research in this thesis could be beneficial to several entertainment production environments such as film, animation and gaming; as well as practical fields such as traffic studies, education and automobile dynamics.

CHAPTER II

RELATED WORK

This thesis builds upon research in related fields including film and animation, automated articulation, virtual and real-world autonomous vehicle design, virtual automobile dynamics and real-world automobile dynamics.

II.1. Film and Animation

There have been several attempts to automate the rigging and animation of automobiles in film and animation [*Cars* 2001; *Grand Theft Auto* 2004; *Speed Racer* 2008]. As technology progresses in the entertainment fields, so do the expectations towards believability and quality of animation. These expectations create strain on the productions of films and animations and continuously push them into the development of new and better techniques. Conversely, several innovative techniques in computer graphics are often produced not in official production environments, but privately.

II.1.1. MadCar

Within the online open source community, Andrey Kozlov privately developed a simulation-based technique for the animation of automobile models for entertainment production. He created an open source program for the commercial software package *3DS Max* [2] called *MadCar* [6]. When implemented, the system simulates physical properties and limitations of an automobile while the user animates that automobile by

“driving” it within the computer generated scene. As the user drives the vehicle, the system records the resulting, simulated animation.

This animation system provides an interesting approach to the problem of believable automobile animation, but it has several production-centric problems. One issue is that the physical simulation system central to the functionality of the program is computationally expensive. While using this program, trying to animate more than one vehicle in a scene at a time becomes noticeably problematic. The refresh rate slows down considerably due to the computational overhead from the system simulating multiple vehicles simultaneously. This limitation hinders the program's usefulness in productions that require the animation of multiple vehicles efficiently.

Also, one cannot manually edit the animation after it is recorded with this technique. In order to make animation changes, a user must rerecord the act of driving the vehicle again from the beginning. This limitation makes this program become more of a “performance” system, rather than a conventional animation system. The repeated iterations that might be required to get the desired motion and the lack of editing capabilities would make this program unreliable and inefficient in a production.

II.1.2. EzCarRig

Also within the online open source community, Laurent Renaud approached the problem of automobile articulation and animation in entertainment production with a technique called *EzCarRig* [11]. The program developed is an automobile animation system that combines key-frame data, NURBS B-spline path-interpolation and a ground detection function. The system includes an automobile rig limited to a user-defined path derived

from a NURBS curve, and a function that keeps the wheels of the automobile adhered to a ground surface geometry.

This technique is effective at producing believable vehicle animation, yet it doesn't provide any automation for rigging articulation in a vehicle model. Another limitation is that the animation of the vehicle is completely dependent on the user-defined interpolation along the path curve. Animating exclusively through path-interpolation differs greatly from conventional animation techniques and may produce some undesired results. Path-based systems decouple positioning and timing: animators create a path in space without regard to timing, and then time character motion along that path [9]. According to Tim Milliron, this technique is awkward for many animators, and makes animation revisions particularly difficult, since altering the path changes its parametric space, which in turn alters the animator's carefully tuned timing [9].

II.1.3. Smart Cars

For Pixar Animation Studios' animated feature film *Cars*, Tim Milliron's team developed an automobile animation system that combined path-based animation techniques with conventional animation techniques [9]. They built specialized procedural-motion systems that allowed animators to simulate physically-plausible motion, but without sacrificing animator control.

Milliron's team also developed a "space-time spline" technique for animating characters along the motion path and timeline. They created a curve through space with time built-in, where each control vertex is defined by the knots of the usual animation channels at a particular frame. This approach helps to alleviate the awkwardness of

switching between path-based animation techniques and conventional animation techniques.

These techniques provided excellent animated performance results. A fault may be found in the lack of definition in the term “physically-plausible motion.” Using path-interpolation for automobile animation may provide physically-plausible results, but these results may not prove to be realistic enough to convince the audience. Accounting for the motion of automobiles, as defined by an automobile's physical limitations, may require more functions than just path-interpolation.

II.2. Automated Articulation

One of the most daunting tasks in computer animation today is appropriately fitting articulation to geometry. A typical digital character requires manual articulation, also known as rigging, to specify its internal skeletal structure and to define how the input motion deforms its surface [3]. Traditionally, rigging has consisted of placing each bone of the skeleton in the approximate location of the desired pivot of rotation [13]. However, as the complexity of the rig grows, this quickly becomes impractical on a large scale.

This process becomes even more tedious when a user is expected to rig similar characters of varying dimensions. There have been attempts to develop functions that automate this process, such as Ilya Baran and Jovan Popovic's program known as *Pinocchio* [3]. Their system when implemented takes a geometric model of a character, analyzes the volumetric characteristics of it through a process called *sphere packing* and then resizes and positions a rigging skeleton to fit inside the character.

This method is similar to Jason Smith and Jeff White's *BlockParty* system [13], developed at Industrial Light & Magic to economize the rigging of many digital characters with similar animation control systems but variable sizes. They developed an approach that also derived the positions needed for rigging and deformations from the geometry, as opposed to defining them by hand. Differing from Baran and Popovic's *Pinocchio* system, Smith and White's technique follows the method of first creating a standardized "volume guide" geometry template for characters. Next, rigging, deformations and muscle elements are created to fit this standard geometry. Following that, the position of each rig element relative to the geometry is recorded. As the volume guide geometry is sculpted to match the high resolution character geometry, the internal rigging elements of the character are accurately re-positioned to track with the geometric changes.

These techniques of automatically positioning articulation elements based on the analysis or manipulation of a character's geometric characteristics have proven to be successful in films [*Transformers* 2007; *Chronicles of Narnia: The Lion, the Witch and the Wardrobe* 2005]. These methods could be used as models of how to approach the procedural rigging of various articulated objects, including automobiles.

II.3. Autonomous Vehicles

II.3.1. Virtual Racing

Hobart Chan approached the issue of automobile animation with the development of a virtual automobile racing simulation system that utilizes a flocking behavior model. One

aspect of his research was the development of a system where the motion of each car is determined by a response to its environment [4]. Rather than explicitly describing the motion of each character individually, the animation for all the cars is automated. Essentially, the system determines the motion of the car based upon elements of the "track" it travels down and the positions of its neighboring cars.

This approach to vehicle animation could be highly effective if expanded into the study of racing dynamics. As for its usefulness for a production in the entertainment industries, it has limitations. Due to the program's lack of direct user input into the animation of the automobiles, animators concerned with creating specific performance requirements would find it limiting. However, it might prove useful for the animation of background vehicles.

One helpful method that can be drawn from this research is its development of position and orientation relationships created directly between the automobile model and the race track. The techniques used in these relationships may be reproduced for other functions such as in the development of ground detection techniques.

II.3.2. Autonomous Real-world Vehicles

In James Massey's research [8] into the automated-driving of real-world vehicles, the problem of automobile motion is approached with the development of a program that helps in the navigation of real-world vehicles based on GPS data. The program implements a series of functions that employ physical calculations involving variables such as longitudinal and latitudinal velocities, steering angle, tire traction, wind resistance, rolling resistance, engine torque, axle torque, and the inertia of the wheels.

Massey's techniques for determining automobile motion are capable of easily being applied to vehicle animation in computer graphics.

For example, in his writings he describes performing many of the navigation functions based on a flat Cartesian plane, no matter how dynamic the actual terrain may be, to simplify the technique [8]. This approach may be expanded upon to more efficiently animate virtual automobiles. Instead of trying to calculate the driving motion of an automobile model over complex terrain, one could use a “zero plane” approach by determining the driving motion of an automobile model on a flat plane and then projecting its position and orientation onto the surface of a terrain.

Also presented in Massey's research are equations depicting the forces and variables to determine the lateral acceleration of the vehicle in motion. Although used for the simulated driving of an actual, real world vehicle, these functions and techniques are helpful when simulating automobile motion in computer generated vehicles. For example, the equations Massey used for calculating lateral acceleration may be easily converted to determine the amount of roll in a virtual vehicle in motion. Then, a procedural function to animate an automobile model's secondary animation can be derived.

II.4. Virtual Automobile Dynamics

Russell Mueller's research into the recreation of an automobile as a physically-accurate computer model demonstrates a simulation approach to automobile articulation and dynamics [10]. To create a complete vehicle dynamics model of a 2002 TAMU FSAE racecar, Mueller used MSC Software's ADAMS/Car program and FSAE templates of the

actual car [10]. The templates divide elements of the vehicle model into sub-models in order to simplify the full vehicle model and to reduce computational requirements, while still providing an accurate representation of the overall vehicle dynamics. Using repeatable templates of key components in a hierarchy structure allows for modularity. This is highly beneficial for the reproduction of complex models and adding variation efficiently. This approach can easily be applied to other methods of automobile rigging.

Mueller also breaks down the entire automobile structure system into subsystems including suspension, steering, powertrain, tires, brakes, and anti-roll bars . Within these subsystems, each moveable part of the automobile is treated as a physically-accurate rigid body. When the virtual automobile is simulated, data is extracted to make accurate predictions of what the real automobile might do under similar driving conditions. This approach is excellent for producing accurate data for real world application, yet it is computationally expensive and inefficient in an entertainment production environment.

II.5. Real-world Automobile Dynamics

II.5.1. Darwinian / Ackermann Steering Geometry

In 1759, Erasmus Darwin designed a steering model to improve the efficiency and safety of horse-drawn carriages. The design was later replicated and patented by Rudolph Ackermann [5]. Darwin's principle states that in a carriage the two front wheels should turn about a center that lies on the (extended) line of the back axle of the carriage [5].

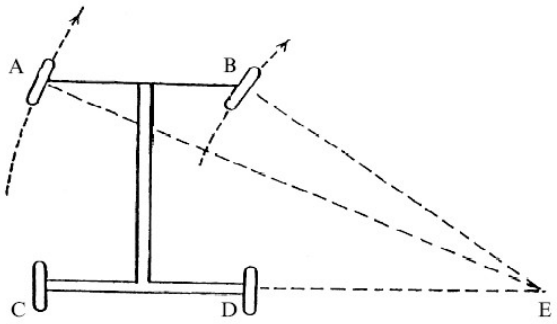

Fig. 1. The Darwinian Criterion

Darwin's optimized steering model paved the way for improved steering geometry in wheeled vehicles (See Figure 1). This basic principle is still the key to modern automobile design and technology today. His efforts to optimize carriage steering helped to advance the field of vehicle safety and performance analysis and encouraged the application of advanced geometric models. Through the implementation of basic trigonometric functions, one could utilize this geometric principle to accurately project the motion patterns of automobile models of any dimension in a virtual environment.

II.5.2. Maneuvering Vehicles

Following in the footsteps of Darwin nearly two hundred and fifty years later, J.C. Alexander and J.H. Maddock take a more in-depth mathematical approach to the analysis and optimization of automobile design with respect to steering. They recreated a mathematical model of a vehicle on rolling wheels [1]. Next, they investigated the general kinematics of such vehicles and developed an Euler-Savary formula relating the center of rotation of the vehicle with the centers of curvature of the trajectories of the axles. They then made a determination of optimal steering for a vehicle turning around a corner, and also how tight a corner a given vehicle can traverse.

Their research not only proves Darwin's research, but also provides an excellent dissection of many of the geometric principles that are acting upon an automobile in motion and the limitations that govern that motion. This research provides a plethora of dynamic functions that can be directly implemented in a virtual environment to determine the motion patterns of automobile models.

CHAPTER III

METHODOLOGY

III.1. Automobile Articulation

Visually, a typical automobile follows the model of four wheels and a chassis. The structural differences between most vehicles can be reduced to variations in chassis and wheel dimensions. When approaching the problem of rigging articulation for automobile models, one focus of this research was the creation of a standard vehicle articulation template. This template would allow a user to automatically rig articulation for nearly any type of automobile of various dimensions.

III.1.1. Rig Template

Rigging is a process used in computer graphics to bridge the gap between modeling and animation. Typically, an animated object is initially constructed using a series of tools to manipulate the virtual geometry into a particular form. This is commonly known as modeling. Next, rigging elements are constructed and linked to the geometry of the object. These elements allow the animator to manipulate the form and position of the geometry of the object.

The use of modularity is highly valued in the process of rigging. It can be achieved through the implementation of standard templates. This is ideal for rapidly constructing multiple models of similar structures when needed. It also provides the opportunity for a user to make changes to one or several models efficiently.

In the program developed for this thesis, a system was created that utilizes customizable automobile rigging templates to allow the user to produce a multitude of automobiles of varying dimensions. Within one of these templates the user may alter the dimensions of the vehicle including chassis length, front and rear axle width, front and rear suspension width, front and rear wheel radius, and front and rear wheel width. With the control of these dimension variables, a user may produce articulation for nearly any type of automobile.

To initiate the customization process the automobile rig is automatically constructed with dimension values provided by a default template. Once the default template is implemented, the user can then change the dimension values to set the appropriate proportions for the automobile model they are rigging. Finally, the user may save those values into a new template so they may reproduce the articulation for that particular automobile model immediately for future use. This process also allows for easy interchangeability between automobile rigs and models while animating.

III.1.2. Fitting Articulation to Geometry

For this program, an automated geometry fitting function was developed to aid in the rigging process. Initially, the user would be expected to provide the function with the geometry of the automobile's four wheels and chassis. Next, the function analyzes the dimensions of each piece of geometry and their positions relative to each other, and resets the dimensions of the automobile rig to fit this geometry appropriately.

This process is achieved by simplifying the structure of the four wheels into the common geometric form of a cylinder. This form can ultimately be broken down into a

few simple geometric equations that can be used to manipulate the position and form of elements in an automobile model's rigging.

III.2. Automated Automobile Animation

Conventional automobile motion follows a pattern based on the automobile's structure as defined in *Darwin's Criterion*. A common automobile follows the structural pattern of two steerable wheels in the front and two non-steerable wheels in the rear. This structure limits maneuverability and produces a predictable motion pattern that can be reproduced with a series of trigonometric functions.

For this thesis, an animation system was developed based on such principles and the motivations of user-friendliness and production efficiency. The system utilizes intuitive “driving” controls and path-interpolation functions in the user interface, Bezier curve functions for animation path construction and manipulation, and techniques reliant on the turning radius and osculating circle principles for the accurate replication of real-world automobile motion limitations and the accurate projection of vehicle position, orientation and steering.

III.2.1. Driving Controls

To ensure efficiency with this program, the number of controls for animating the automobile model has been kept to a minimum. To set the automobile's initial layout position and orientation, or to parent the model to another object in the scene, a control object called the *Root Control* is used as the model's root node. A control object is a node built into a model's rigging that allows the animator to manipulate the model through

direct manipulation of the control object's attributes; including position, orientation and scale. In common rigging practices, the root node is the top level control object in a model's articulation hierarchy.

To create the “driving” animation of the automobile model, a control object called the *Driver Control* is used to set the initial vehicle position and orientation, and to manipulate the path curve that the model follows through the user-defined manipulation of the *Driver Control's* position and orientation. Additional animatable variables are added to the *Driver Control* to allow a user to manipulate the path curve, set parameters in the simulation, set the forward or reverse direction for driving, edit secondary animation values and set dimensions of the automobile rig.

III.2.2. Path Step

Using Milliron's “space-time spline” technique as a model, a function called the *Path Step* was developed to combine path-interpolation animation with conventional animation. This technique is achieved by utilizing position and rotation animation data from the *Driver Control* object.

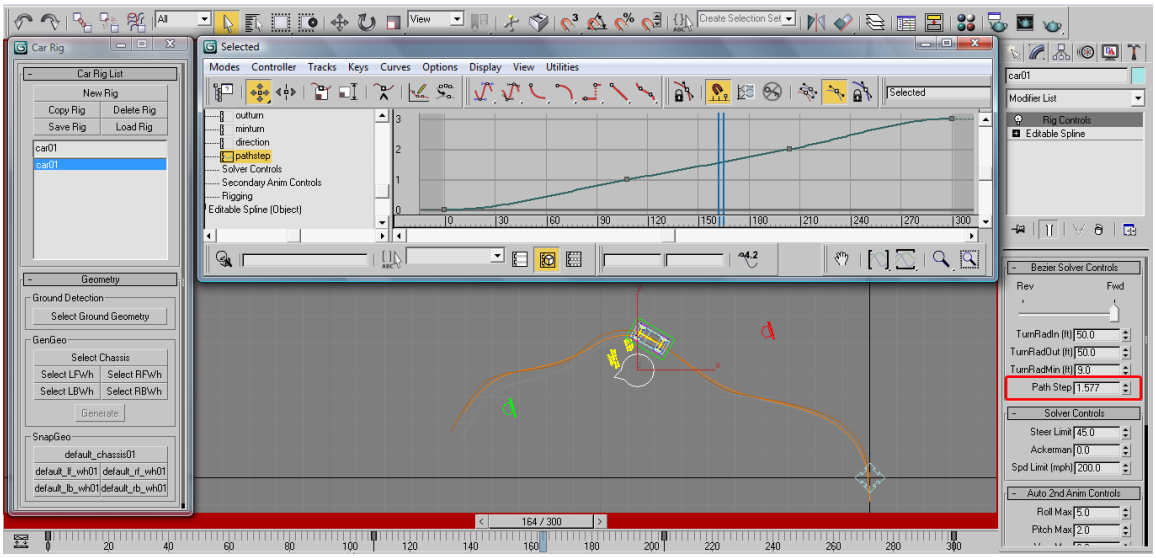

Fig. 2. Path Step

As the user animates the position and rotation of the *Driver Control* object, a single animation curve is built with key frame values corresponding in time to key frame values in the *Driver Control* object's position and rotation animation data (See Figure 2). The values of the *Path Step* curve determine the parametric distance the automobile model should travel along the animation path curve between keyed time values. Essentially, each keyed value of the *Driver Control* represents a normalized start and end value of a section in the animation curve. These individual animation curve sections are combined together to form a single animation curve. The user may manipulate the shape of this curve to modify position values of the vehicle model over time.

III.2.3. Bezier Curve

Pierre Bezier was a French engineer with the Renault car company who set out in the early 1960's to develop a curve formulation which would lend itself to shape design [12].

He utilized a geometric algorithm developed by Paul de Casteljau to create editable parametric curves to aid in automobile body design. These curves are still used today in nearly all fields of computer graphics and design due to their precision and user-friendly nature.

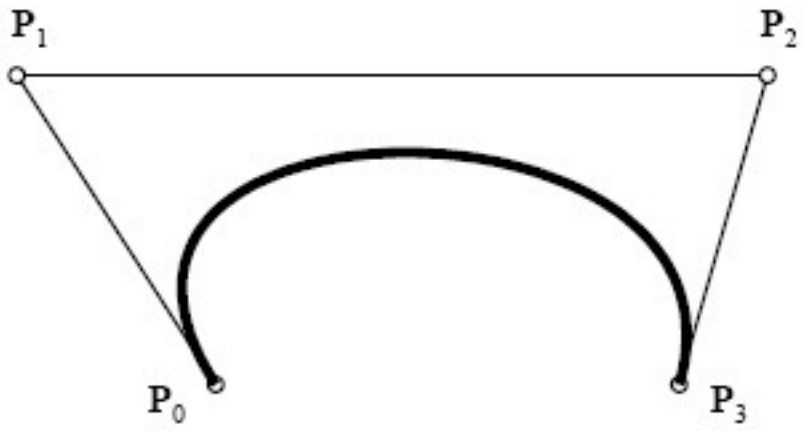

Fig. 3. Bezier Curve

For the program developed in this research, Bezier curves were implemented for creating the initial animation paths and trajectory curves of the animated automobile models. These parametric curves rely on "control points" to determine their structure and curvature (See Figure 3). This is ideal for manipulating with user defined control objects in a virtual environment.

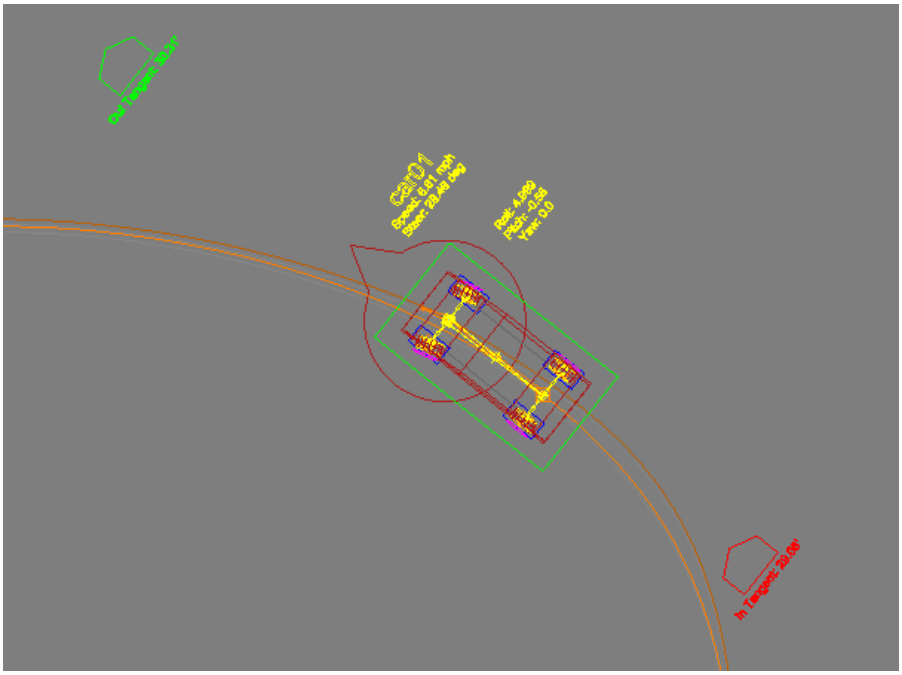

Fig. 4. Bezier Controls

As the user animates the *Driver Control* in the program, third degree Bezier curves are constructed between each keyed position of the *Driver Control* in the virtual environment. A *Bez In Control* object and a *Bez Out Control* object are used to set the positions for the *In Tangent* and O*ut Tangent* points of each knot at the ends of each Bezier curve. To maintain continuity between each Bezier Curve, the *Bez In Control* and the *Bez Out Control* positions are limited to positive and negative values along the local Y-Axis of the *Driver Control* object (See Figure 4). The individual Bezier curves are linked together to create the entire animation path that the automobile model initially follows along.

III.2.4. Turning Radius

The radius of the smallest circular turn a vehicle is capable of making is commonly known as the turning radius, or turning circle. In accordance with *Darwin's Criterion*, the two front wheels of a turning automobile will follow a circular path around a center point that lies on the extended line of the back axle of the vehicle.

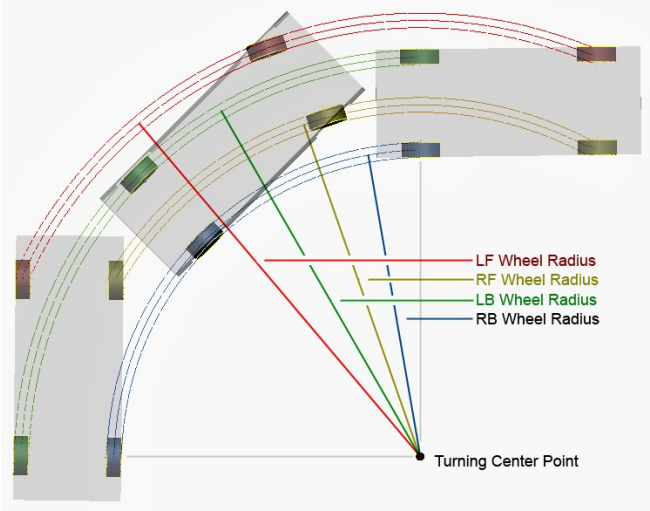

Fig. 5. Turning Radius

As the automobile is in motion, the paths of all four wheels will draw curves equivalent to portions of circles at varying radii from the aforementioned center point (See Figure 5). The smallest circle drawn by one of the four wheels (typically the rear interior wheel on a modern automobile) is where the turning radius, or turning circle, variable is derived.

In a typical automobile, the turning radius is derived from the structural and mechanical limitations of the vehicle's steering mechanisms. If an automobile did not

have steering limitations, according to *Darwin's Criterion*, the minimum turning radius would be equivalent to the distance between the front and rear wheel axles.

With this principle implemented in conjunction with trigonometric functions, one could ascertain the position of a turning center point projected alongside an automobile at any given steering angle. For example, if a vehicle were turning at its maximum steering angle, the “live turning radius” of the vehicle would be equivalent to its turning radius limit. Conversely, as a vehicle's steering angle approaches zero, or the equivalent to driving perfectly straight, its “live turning radius” approaches infinity.

A series of functions based on this principle were developed for this project. They were implement to allow the user to set a minimum turning radius limit for each vehicle model as it is animated. This ultimately helps to more realistically define the path the automobile model will follow.

III.2.5. Osculating Circle

At any given position along a smooth curve, the curvature of a point on that curve can be matched to the curvature of a tangential circle. This geometric principle is known as the *Osculating Circle* (See Figure 6). When implemented, position and orientation data for points on any smooth curve can be calculated using circle geometry models and trigonometric functions.

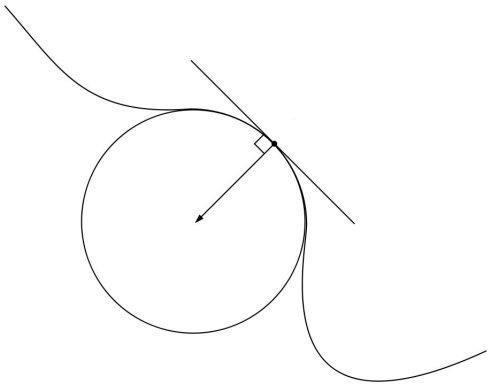

Fig. 6. Osculating Circle

This principle was used in this project in conjunction with the turning radius principle to correct a vehicle's animation path. The system would take in the user-defined, turning radius variable and apply it to the *Osculating Circle* principle to re-project an animation path that respected the automobile's steering limitations. As a result, the vehicle model would have more realistic automobile motion.

III.3. Automated Ground Detection

One of the major contributions to the believable animation of a computer generated character is its dynamic response to other elements in the virtual environment. For example, if a character model were not able to respond appropriately to changes in elevation as it walked across a ground surface, the believability of the animation would become questionable.

For this thesis project, an automated ground detection function was implemented in the program to maintain the vehicle's believable adherence to ground surface geometry while in motion. As a vehicle model is animated driving over a surface, rays are cast down from the center of each of the vehicles wheels to determine the distance to the

surface geometry. A comparison is made between the projected distance and the radius of each wheel. An elevation and orientation value for the automobile is calculated by averaging the values of all four wheel elevations. The automobile is then translated and rotated appropriately to maintain believable motion as it traverses the geometric surface.

III.4. Secondary Animation

Secondary animation is motion that results directly from other animation [7]. Examples may be seen in the jiggling of body mass as a character walks or the swaying of an antenna as an automobile turns. A secondary action is always kept subordinate to the primary action [7]. This motion applies not only to soft, deforming objects, but to rigid articulated objects including automobiles with suspension systems.

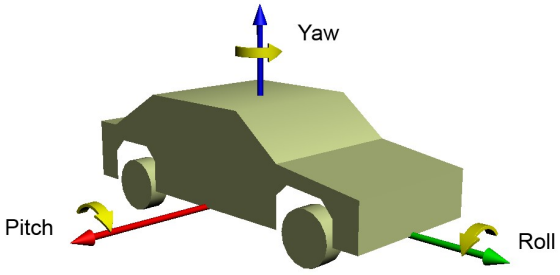

Fig. 7. Secondary Animation

For the automated animation system in this project, a series of functions were used to determine the automobile's rolling, pitching and yawing motion (See Figure 7) based on the automobile model's turning and velocity values as it is animated. The implemented rolling and yawing functions required the use of centripetal force calculations. One of the major factors needed in calculating any force is the mass of the object in motion.

$$Force = ma = \mathrm{d}(mv) / \mathrm{d}t$$

$$Centripetal\ Force = ma_c = mv^2 / r$$

To bypass the necessity for an animator to input the automobile's mass, the mass value is completely removed from the calculations. The system calculates the modified secondary motion values throughout the entire timeline and records the maximum pitch, roll and yaw value. These maximum values are used to derive normalized values for the secondary motion per frame. The normalized values are then multiplied by a user-defined pitch, roll and yaw rotation angle value to allow the user to customize the amount of rotation that will act upon the vehicle chassis at any given time in the animation. As a result, the input of the automobile's mass is no longer required.

Another variable that is usually required in calculating secondary motion of an automobile is the spring coefficients of the suspension system. Due to the user-defined nature of this technique and the focus on the velocity and turning values of the vehicle, the spring coefficients are no longer necessary.

III.5. Darwinian / Ackermann Steering Geometry

Since Darwin's steering geometry, also known as Ackermann steering geometry, is such a common attribute of modern automobiles, it seemed appropriate to include it in the rigging model. And since it is an attribute that changes in value between automobile designs, it also seemed appropriate to make it a value the user may edit in the customization process.

For this project, a series of functions were created for the animation portion of the program that utilize trigonometric equations based on *Darwin's Criterion* (see Figure 1) to replicate accurate Ackermann steering geometry.

Initially, when the automobile model is animated and the motion is calculated in the simulation function, the front wheels of the model are set to be parallel to an acting simulation node to capture the simulated steering values. When the Ackermann function is applied to the front wheel rigging, trigonometric equations offset the steering values of each of the front wheels to match the angles Darwin specified for optimal steering. This produces the desired results as defined by the animator in the user-defined *Ackermann Variable.*

CHAPTER IV

IMPLEMENTATION

The following section discusses aspects of the implementation including the software and programming language used for the implementation environment and the functions implemented in the initialization of the program, the automated articulation, the automated animation, automated ground detection and automated automobile dynamics.

IV.1. Implementation Environment

For the development of this automated articulation and animation tool, a commercial 3D modeling, animation and rendering software package called *3DS Max* [2] was used. This program was chosen due to the accessibility of its interfaces, the flexibility of its programming language, its robustness as a solid, reliable program, and its popularity among the gaming, film, and animation industries.

The virtual objects, functions, and simulations necessary for the development of this tool were written in *3DS Max's* native programming language *Maxscript*. The user-interfaces and control objects were also implemented through *Maxscript*. These include a series of control attributes for each rig and a floating interface window for the management of the automobile model rig list. Ultimately, the program is executed as a loadable script in *3DS Max*.

IV.2. Automated Articulation

IV.2.1. Initialization

Upon the user's initialization of the program script, the system creates an array of all the elements in the current scene and runs a filter on them, searching for an included string ("*AutoRig")* flag within each element's attributes. Once the program has identified and collected all the appropriate rig elements containing the search flag attribute, the system reconstructs the internal connections and variables of each automobile rig, and adds the newly reconstructed rigs to the main *Rig List* array. From here, the user may continue to edit the articulation and animation of the automobile models.

IV.2.2. Rig Building

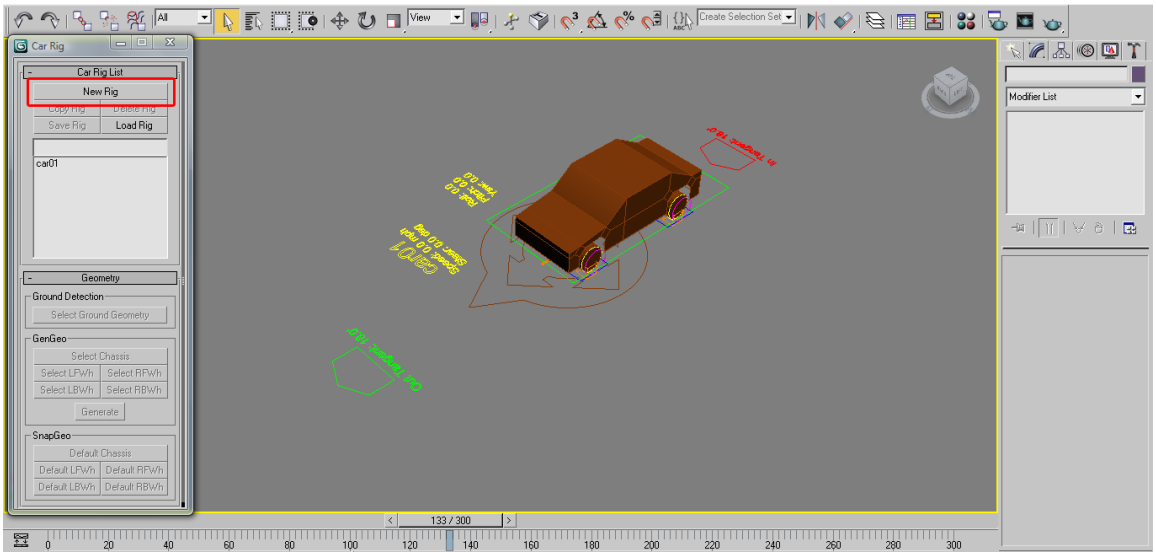

Fig. 8. New Rig

When the user clicks the *New Rig* button in the *Car Rig List* interface (See Figure 8), a new, default automobile rig is added to the scene *Rig List* array. Following that, a new

rigging template is constructed to store the rig's variables and attributes with initial default values. Then, the *buildrig* function is called to construct the elements of the rig in the virtual scene.

When the *buildrig* function is called, it constructs the control objects, NURBS curve paths, display texts, simulation rigging, automobile rigging, ground detection rigging and default geometry for the new automobile rig. Finally, the dimensions of the new rig are altered to match the values stored in the rigging template.

IV.2.3. Geometry Fitting

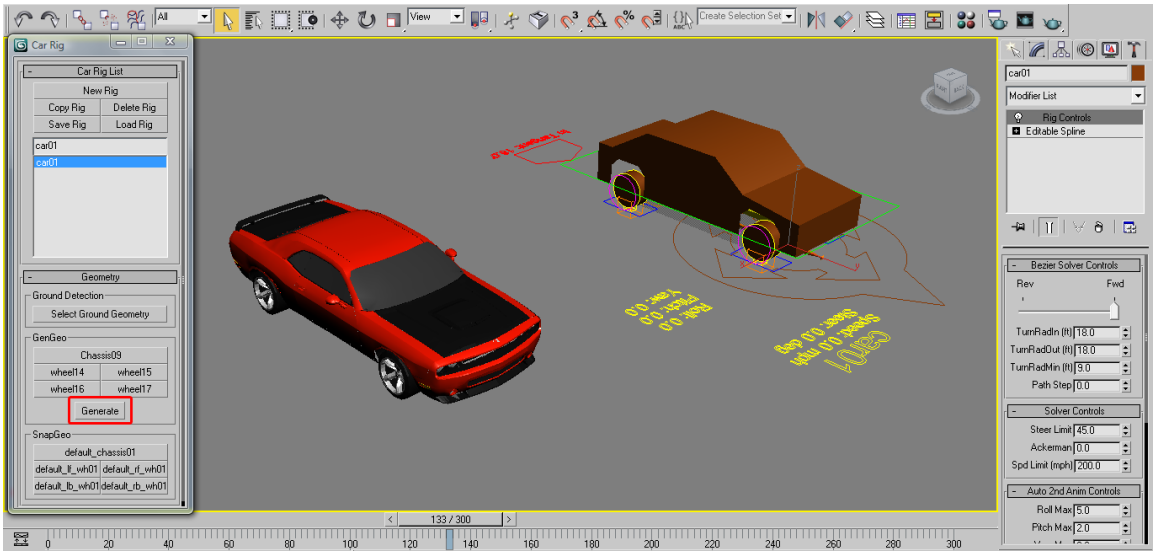

Fig. 9. GenGeo

After the user selects the appropriate chassis and wheel geometry for an automobile rig, and clicks the *Generate* button in the *GenGeo* section of the *Geometry* user interface (See Figure 9), the newly selected geometry is set as the automobile's temporary geometry and the *GenGeo* function is called.

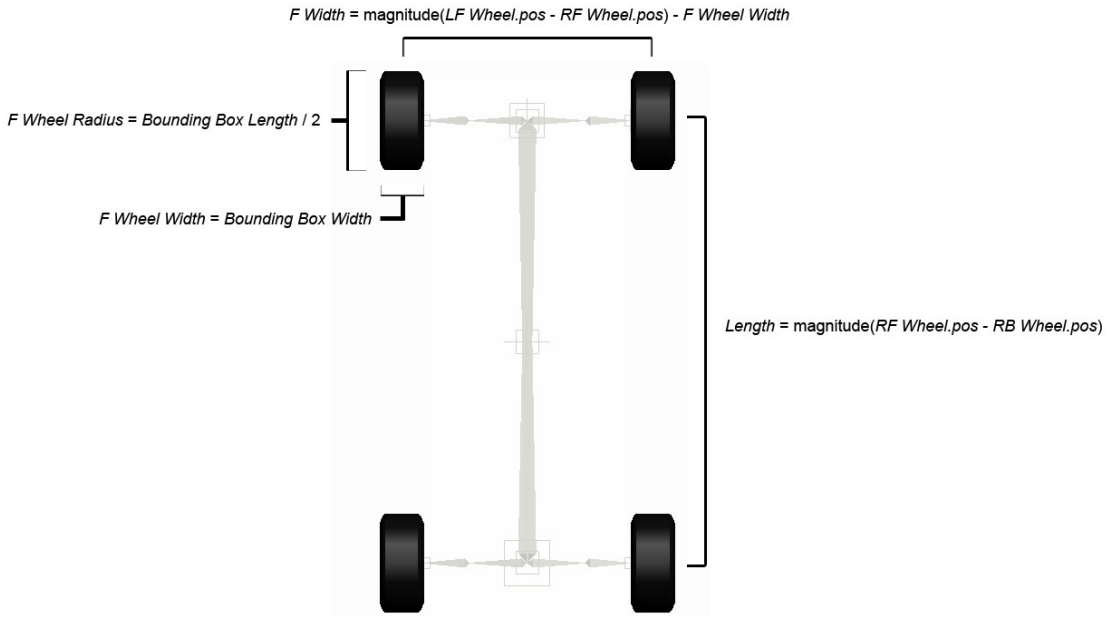

Fig. 10. Wheel Measurements

When the *GenGeo* function is called, it first calculates the dimensions of the target automobile model geometry. This includes extrapolating the length and width of the automobile chassis rigging. The front axle width is calculated by measuring the distance between the target front wheel geometry objects. To calculate the rear axle width, the system measures the distance between the target rear wheel geometry objects. To calculate the length of the chassis rigging, the distance between the target front and rear wheel geometry objects is measured (See Figure 10).

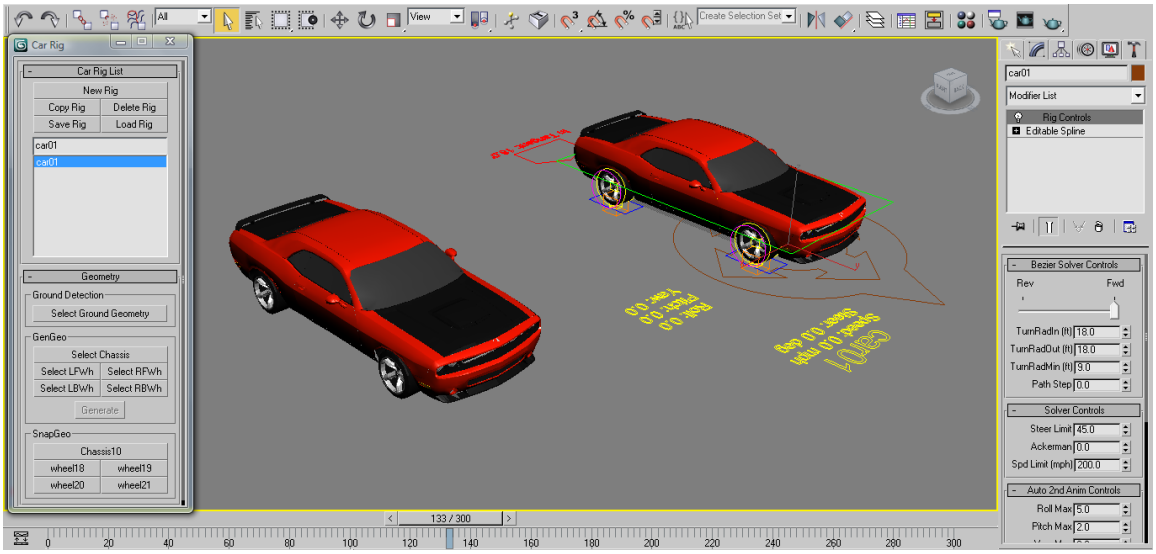

Fig. 11. GenGeo Complete

After the dimensions of the chassis articulation are calculated, *GenGeo* calls a series of *SnapGeo* functions. These functions calculate and set the widths and radii of the wheel articulation by analyzing the vertex position data of the new wheel geometry using *3DS Max's BoundingBox* function. The *BoundingBox* function is a tool common to several commercial 3D modeling, animation and rendering software packages. This tool extrapolates the length, width and height of an object's geometry with respect to a user-defined coordinate system. *SnapGeo* then repositions and reorients the new wheel and chassis geometry objects to the automobile rigging based on their local offset values (See Figure 11).

IV.2.4 Save / Load Rig

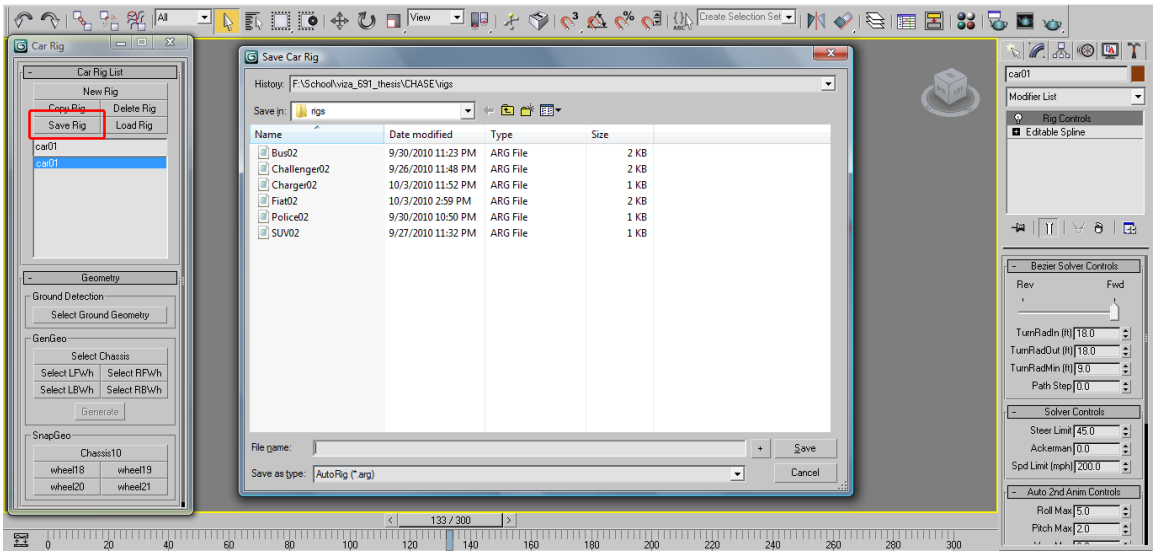

Fig. 12. Save Rig

When the user clicks the *Save Rig* button in the *Car Rig List* interface, the system prompts the user to save the template data of the currently selected automobile rig to a custom text file (*.arg*) and location on the user's computer (See Figure 12). After the user chooses the name and location of the *.arg* file, the system writes the template data to it. Following that, the system also saves the geometry data of the automobile model as a separate *.max* file in the same directory with the same name as the *.arg* file, but with an additional "_geo" suffix.

When the user clicks the *Load Rig* button in the *Car Rig List* interface, the system prompts the user to load an automobile rig file (*.arg*) from a user-defined location. When the rig file is loaded, the *buildrig* function is called again and a new rig is built with the dimension values described in the rig file. Then, the geometry stored in the

corresponding *.max* file is loaded into the current scene and reattached to the rig using the *SnapGeo* functions.

IV.3. Automated Animation

IV.3.1. Callback Functions

One of the key elements to animating the automobile in the program is the use of *3DS Max's Callback* function. Whenever an object in the scene is selected and transformed, the *Callback* tool searches through a list of the objects currently selected and determines if the selection is a control object in an automobile rig or not.

If the selection is determined to be a *Driving Control* object and an attribute of that object is modified during animation the system implements the *Solver* function. The *Solver* function is essentially a list of all the procedural functions the animation system calls to update the animation data. These include a series of Bezier curve, simulation, ground detection, secondary animation, and Ackermann solving functions.

IV.3.2. Bezier Curve Functions

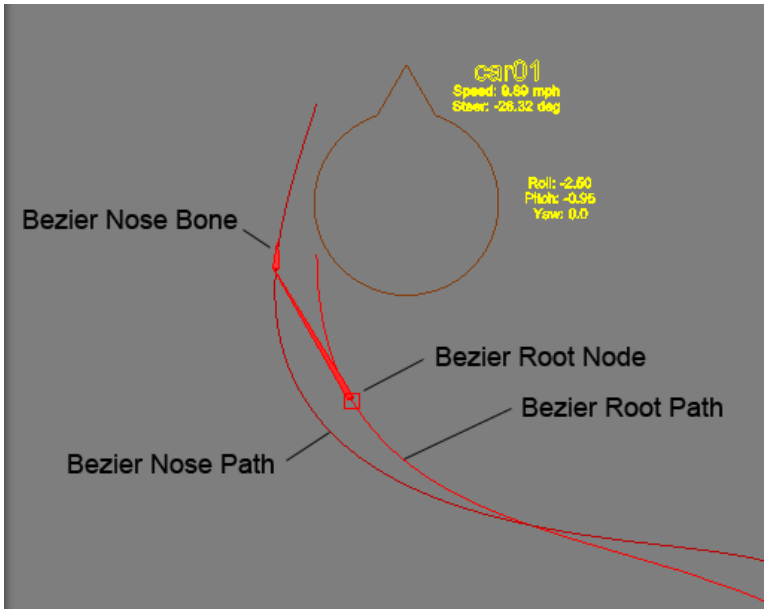

Fig. 13. Bezier Rig

In the *Bezier* section of the *Solver* function, a simplified vehicle model called the *Bezier Rig,* is animated along a motion path defined by Bezier curves constructed from the position and rotation animation data of the *Driver Control* object (See Figure 13).

The *Bezier Root Path* function constructs the Bezier curves of the initial animation path curves located between the key-framed *Driver Control* positions. It takes in the keyed position data of the *Driver Control* to set the Bezier control knot positions of each curve. It then takes in the *Bez In Control* and the *Bez Out Control* position values to define the *In Tangent* and *Out Tangent* values of each control knot, and thus defining the shape of path curve. The *Bezier Rotation / Position* function determines the position and rotation of the *Root Node* of the *Bezier Rig.* To determine the position of the node, the system implements *3DS Max's Path Interpolation* function. The function derives the

position value on the path curve based on its parametric position according to the user-defined *Path Step* value. To determine the rotation of the node, the system implements *3DS Max's Path Tangent* function. This function produces a tangent vector value of the path curve based on its parametric position according to the *Path Step* value. Using trigonometric calculations, a rotation value about the Z-Axis is determined for the *Bezier Root Node* using the given vector.

To determine the *Bezier Rig's* steering values, the *Bezier Nose Path* function generates a curve based on the *Bezier Nose Bone's* position per frame. Then, the *Bezier Steer* function generates steering data based on that curve using the same *Path Tangent* function just like the *Bezier Rotation / Position* function before.

IV.3.3 Simulation Curve Functions

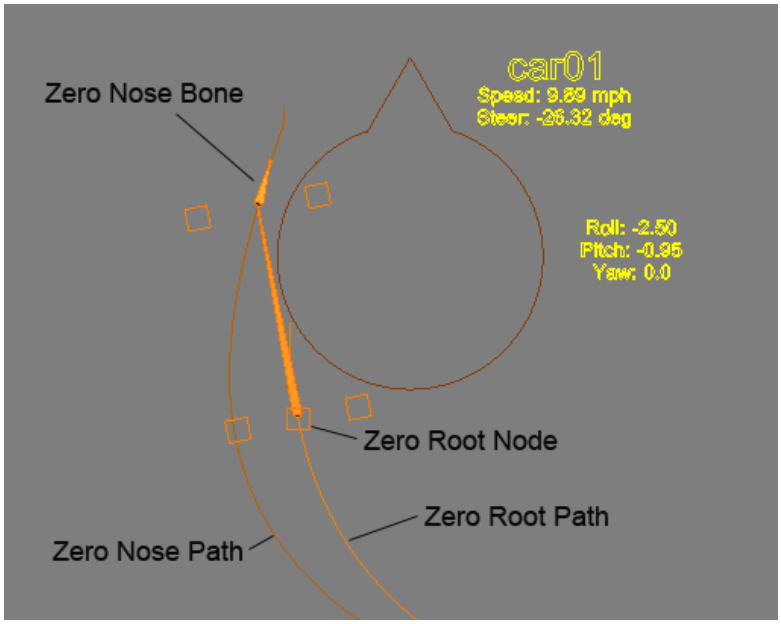

Fig. 14. Zero Rig

The next portion of the *Solver* consists of a series of functions that analyze the curvature of the *Bezier Path Curve* and construct a new animation path curve shaped with respect to the user-defined automobile speed and steering limitations. These functions redefine the automobile model's animation with another simplified vehicle model known as the *Zero Rig* (See Figure 14). The *Zero Rig* follows the newly constructed animation path and provides a more accurate representation of automobile driving motion than the *Bezier Rig*. It is called the *Zero Rig* because it is the last rig in the process to be calculated through the "Zero Plane" ideal as described by Massey.

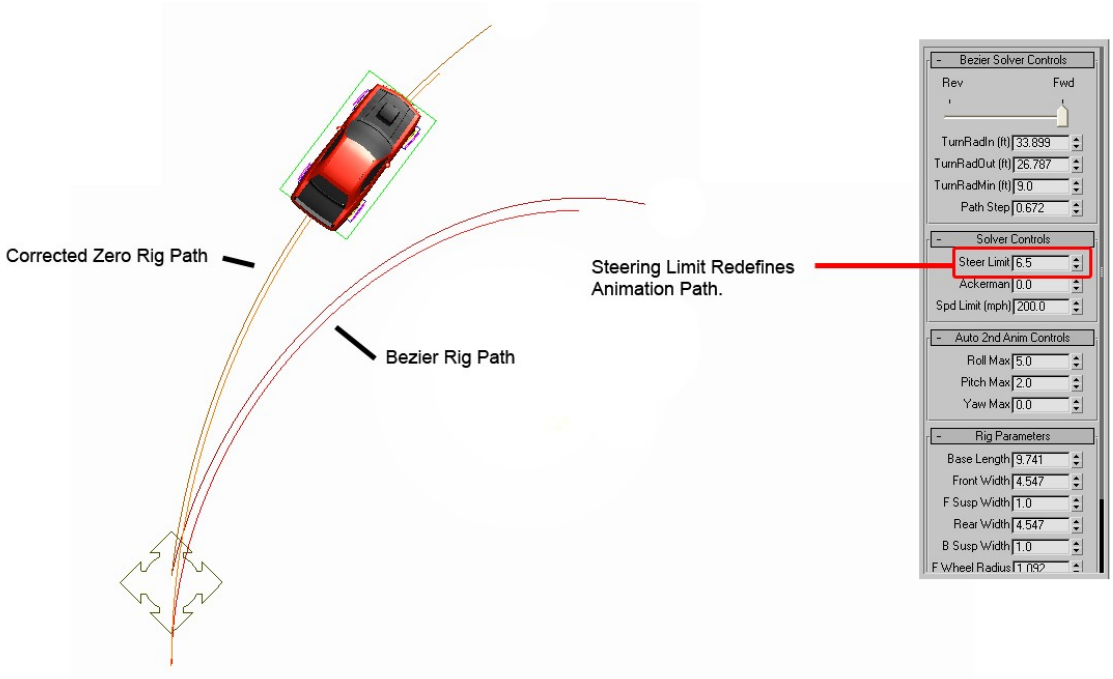

Fig. 15. Redefining the Animation Path

To construct the animation path curve for the *Zero Rig,* the *Zero Root Path* function evaluates the animation data of the *Bezier Rig* to define the automobile model's

speed and steering angle. Those values are then adjusted appropriately with respect to the user-defined limitations (See Figure 15). First, it records the changes in position and rotation of the *Bezier Rig Root* node per frame. This data is then piped through an equation based on the *Osculating Circle* (See Figure 6) to get the immediate turning radius of the automobile model per frame.

$$radius = (dpos / 2) / \sin(drot / 2)$$

Once the radius is calculated, the steering angle of the car can be computed using an equation based on the *Vehicle Turning Radius* geometric model (See Figure 5).

$$angle = \text{atan}(chassis\ length / radius)$$

The steering angle is then compared to the user-defined steering limit and altered to respect the limitation. The function then calculates the speed of the car from the change in position value per frame. The speed is compared to the user-defined speed limit and altered to respect that limitation.

The *Zero Position* function sets the position of the *Zero Root Node* using the *Path Interpolation* function in tandem with the current *Path Step* value. The *Zero Rotation* function sets the rotation of the *Zero Root Node* using *3DS Max's Path Tangent* function with the current *Path Step* value.

To determine the *Zero Rig's* steering values, the *Zero Nose Path* function generates a curve based on the *Zero Nose Bone's* position per frame. Then, the *Zero Steer* function generates steering data based on that curve using *3DS Max's Path Tangent* function.

IV.4. Automated Ground Detection

IV.4.1. Zero Plane Projection

The rigging elements that determine the “driving” animation of the automobile model include the *Bezier Rig* and the *Zero Rig,* and are calculated on a perfectly flat plane known as the “Zero Plane”. The portion of the vehicle model pertaining to the ground detection of the automobile rig is called the *GD Rig.* It is projected along the Z-Axis from the *Zero Rig* to the user-defined *Ground Surface Geometry* using the *Ground Detection* function. This function determines the automobile model's vertical position, local x and y orientation relative to the ground surface, and vertical wheel position relative to the automobile's chassis.

To maintain the accuracy of the steering orientation relative to the “Zero Plane”, *3DS Max's Look-At Constraint* is implemented. As the orientation of the *GD Rig* changes due to the motion over the ground surface geometry, the steering remains constant and accurately projected.

IV.4.2. Raycasting

To produce the effect of ground detection, raycasting and polygonal face functions are implemented in the *Ground Detection* function. As the *Driver Control* object and the automobile model are animated, a ray is cast from the X and Y position value of each wheel node and the Z position value of the *Driver Control* object. The rays are cast in the negative direction along the World Z-Axis. *3DS Max*'s *Intersect Ray* function is used to

determine the point in space where the rays intersect with the user-defined *Ground Surface Geometry.*

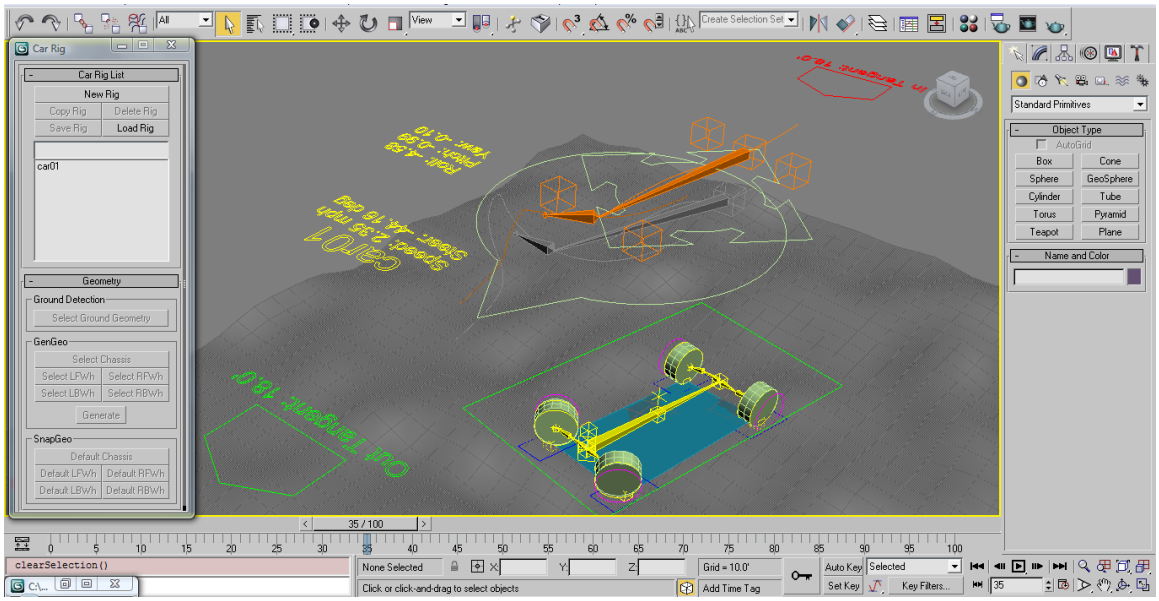

Fig. 16. Ground Detection

The four vertex point positions of a polygonal plane called the *Ground Detection Net* are set to those intersection points derived from the ray functions (See Figure 16). Essentially, a plane with dimensions corresponding to those of the automobile rig is projected onto the ground surface geometry using ray intersect functions for each wheel.

3DS Max's Face Normal function is used to determine the normal vector value of the *Ground Detection Net's* one polygonal face. This normal vector is then used to determine what the orientation of the automobile *Chassis* should be relative to the ground surface geometry. *3DS Max's Face Center* function is used to determine the position at the center of the *Ground Detection Net*. This position value is then used to determine

what the vertical position of the *Chassis* should be relative to the ground surface geometry. Once the *Chassis* is positioned and oriented, the wheels of the automobile rig are translated in their local Z-Axis to positions corresponding to the vertex points on the *Ground Detection Net.*

IV.5. Automated Automobile Dynamics

IV.5.1. Secondary Animation

A series of equations based on the automobile model's change in rotation and velocity are used to determine the vehicle model's rolling, pitching and yawing motion.

To calculate the automobile model's secondary rotation values at a given time, the *Secondary Animation* function first reads the pitch, roll and yaw values over the entire animation timeline and finds the maximum values of each to be used later to produce normalized pitch, roll and yaw values.

$$pitchmax = |velocity|$$

$$rollmax = |velocity^2 / turning\ radius|$$

$$yawmax = |velocity^2 / turning\ radius|$$

Equations are then implemented to ascertain the *Normalized Pitch, Roll* and *Yaw* value of the automobile rig per frame.

$$normalized\ pitch = velocity \,/\, pitchmax$$

$$normalized\ roll = velocity^2 \,/\, turning\ radius \,/\, rollmax$$

$$normalized\ yaw = velocity^2 \,/\, turning\ radius \,/\, rollmax$$

Then, the *Pitch, Roll* and *Yaw* values of the automobile rig are derrived by combining the *Normalized Pitch, Roll* and *Yaw* values with the user-defined *Pitch, Roll* and *Yaw Limit*.

$$pitch = npitch \times pitch\ limit$$

$$roll = nroll \times roll\ limit$$

$$yaw = nyaw \times yaw\ limit$$

These values are then implemented into the rotation values of the *Secondary Root* node of the automobile rig to produce the effect of secondary animation.

IV.5.2. Ackermann Geometry

To calculate the effect of *Darwinian / Ackermann Geometry,* a series of equations based on *Darwin's Criterion* (See Figure 1) are implemented in the *Ackermann* function. First, the immediate turning radius of the car is calculated.

$$radius = chassis\ length \,/\, \tan(steering\ angle)$$

Then, the turning radius of each of the front wheels is calculated.

$$left\ radius = radius - (chassis\ width \,/\, 2)$$
$$right\ radius = radius + (chassis\ width \,/\, 2)$$

Next, the *Darwinian / Ackermann* rotation values for both of the front wheels are calculated. The user-defined *Ackermann* value in the *Driver Control* attributes is used to determine the amount of *Ackermann* effect in the equation.

$$rotation = (\text{atan}(chassis\ length / radius) - steering\ angle) \times acker$$

This rotation value is then used to offset the initial rotation value of the front wheels and give it the desired effect of optimized steering.

CHAPTER V

RESULTS

This vehicle articulation and animation tool is capable of creating believable animation of automobiles. To test the program a short animation called *Chase! a*nd a series of demonstration videos were created with the utilization of this tool. Within the short animation, several features were used to challenge the system including a variety of automobile designs to test the range of the automated rigging function, dynamic animation paths to test the navigability of the animation tools and complex terrain to test the automated ground detection function.

In the demonstration videos, key aspects of the automated vehicle rigging and animation system are highlighted, including automated rig creation, automated geometry fitting, editable rig parameters, automated Ackermann geometry, instant geometry swapping, Bezier path animation, speed and steering limit-based animation, automated secondary animation and automated ground detection.

V.1. Specifications

The computer used to implement this program in the animation sequence and the demonstration videos contained an AMD Athlon II Quad Core processor at a rated processing speed of 2.90 GHz and 6.00 GB of RAM.

V.2. Animation Sequence

V.2.1. Articulation

For the articulation portion in the production of the animation sequence, rigging was created for automobile models in the sequence using the automated articulation system in this tool. The program was capable of automatically creating articulation for 5 unique automobile models including a 2010 Dodge Challenger, a 2009 Dodge Charger, a 1957 Fiat 500, a 2009 Toyota FJ Cruiser and a city bus (see Figure 17).

Fig. 17. Animation Sequence Automobiles

Approximately 15 seconds were required for the system to automatically generate an animation-ready vehicle model of each unique automobile type. To begin the process, 1.16 seconds were required for the system to automatically generate a new *Template Rig*

following the user clicking the *New Rig* button in the *Car Rig List* interface. It then required 0.12 seconds for the *GenGeo* function to automatically fit the articulation of the *Template Rig* to the geometry of that unique vehicle model following the user clicking the *Generate* button in the *GenGeo* section of the *Geometry* interface. Approximately 13.72 seconds were required for human interaction between the rigging functions, including the selection of the geometry objects for the *GenGeo* function. For the duplication of the automobile models, each unique automobile rig was saved to a user-defined location and then copies of those rigs were loaded into the scene as needed. In total, 21 automobile models were constructed and managed using the various articulation and model management tools of the automated rigging system.

V.2.2. Animation

Fig. 18. Chase!

For the animation portion of this production, the automated animation system was capable of creating the vehicle motions for the various automobile models necessary for the animation sequence (See Figure 18). The total length of time for the *Chase!* animation sequence was 12.5 seconds at 24 frames per second (300 frames). The system was able to allow for the simultaneous animation of 21 automobile models within the sequence. Of those vehicle models, 6 were animated over complex terrain accurately using the Ground Detection system. The animation system required 0.41 seconds to calculate 300 frames of animation for one vehicle model not using the Ground Detection system and 4.12 seconds to calculate 300 frames of animation for one vehicle model using the Ground Detection system.

With the inclusion of time required for human interaction with the articulation and animation tools, and the continuous editing and adjustment of the vehicle actions and placements; the articulation and animation of the *Chase!* sequence required approximately 10.5 hours of work.

CHAPTER VI

FUTURE WORK AND CONCLUSION

VI.1. Future Work

Future work on this project could address additional attributes of the automobile simulation such as mass and force calculations. The addition of mass and force calculations could help provide a more realistic approach to automobile animation. The user could input real-world mass and friction values for the automobile chassis and wheels, and the system could animate the automobile more accurately with respect to actual real-world forces such as inertia, torque, and traction.

VI.2. Conclusion

The tool developed in this thesis solves many problems found in the articulation and animation of computer generated automobiles. It does so by enabling the user to rig and animate a variety of vehicles quickly and realistically. This is aided by simple and user-friendly interfaces that provide vast amounts of control and customization in the automobile models being produced. The emphasis on production efficiency through the ease of interface and the rapid generation of materials exemplified in this project give it immediate value.

This tool is not a physically-accurate model of vehicle behavior. It was developed with the expressed motivation to be a production tool with a primary focus on the importance of producing convincing imagery of automobiles in motion for film and animation.

This tool may be used as a model for the development of similar techniques used for the optimized and automated rigging and animation of other types of vehicles or characters. It also may be used if applicable in fields outside of entertainment, including those in automobile design, traffic study and roadway optimization.

REFERENCES

[1] J. Alexander and J. Maddocks, “On the Maneuvering of Vehicles,” *SIAM Journal on Applied Mathematics,* vol. 48, no. 1, pp. 38-51, 1988.

[2] Autodesk 3DS Max 2009, http://usa.autodesk.com, 2009.

[3] I. Baran and J. Popovic, “Automatic Rigging and Animation of 3D Characters,” *ACM Transactions on Graphics*, vol. 26, no. 3, July 2007.

[4] H. Chan, *Vehicular Racing Simulation: A Mel Scripting Approach,* M.S. Thesis, Texas A&M University, 2004.

[5] D. King-Hele, *Erasmus Darwin's Improved Design for Steering Carriages,* London: The Royal Society, 2002.

[6] A Kozlov, “MadCar,” *Max Underground*, 2010. http://www.maxunderground.com/archives/3769_madcar_v1_0.html

[7] J. Lasseter, “Principles of Traditional Animation Applied to 3D Computer Animation*,” ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 35-44, 1987.

[8] J. Massey, *Control and Waypoint Navigation of an Autonomous Ground Vehicle,* M.S.Thesis, Texas A&M University, 2006.

[9] T. Milliron and F. Behmaram-Mosavat, “Smart Cars: Driving the Characters in *Cars*,” *SIGGRAPH '06, Sketches, no. 116*, SIGGRAPH '06, 2006.

[10] R. Mueller, *Full Vehicle Dynamics Model of a Formula SAE Racecar Using ADAMS/Car,* M.S. Thesis, Texas A&M University, 2005.

[11] L. Renaud, “EzCarRig,” *Loran*, 2009. http://laurent.renaud.free.fr/ezcarrig.html

[12] T. Sederberg, class lecture for “Bezier Curves,” Brigham Young University, January 6, 2003.

[13] J. Smith and J. White, “BlockParty: Modular Rigging Encoded in a Geometric Volume,” *SIGGRAPH '06, Sketches, no. 115,* SIGGRAPH '06, 2006.

VITA

Christopher Corey Griffin

griffin@paradoxclock.com

Education

M.S. in Visualization, Texas A&M University, December 2010

B.E.D in Environmental Design, Texas A&M University, May 2007

Employment

Resident Character TD, Pixar Animation Studios

January 2010 – January 2011

Address

Department of Visualization
Texas A&M University
C108 Langford Center
3137 TAMU
College Station, TX 77843-3137
c/o Tim McLaughlin