Modifying Road and Vehicle Databases at Hills and Curves

A Thesis Presented

by

Govindaraaj Mudumbai

to

The Department of Mechanical and Industrial Engineering

in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Systems Engineering

in the field of
Engineering Software Design

Northeastern University
Boston, Massachusetts

December 2008
Abstract

An important aspect of virtual environments is the design of roads, and vehicular movements. Their smoothness helps define the face-validity of the virtual environment. This research presents procedures used to create smooth roads and smooth vehicular paths.

The building block of this face-validation is the algorithm used for creating curves. These algorithms take in the start, end, and radius of the circle formed by the arc, and number of segments (n). It divides the angle subtended by the arc into n. For each segment, the four corners are calculated and sent to a road building function. The more the number of segments the smoother the road is.

Once the roads are smooth on the curves and hills, the next step is to make the vehicles traverse smoothly on the roads. The path of the vehicles is rendered using a different curve algorithm that takes in the start, end, and center of the circle formed by the arc. To find the center point of the circle formed by the arc (the path to be taken by the vehicle), we have to plot at-least three points of the arc in Autocad. After fillet operations on the arc, the properties of the perfectly fitting arc have the center point (x, y and z coordinates) of the arc. After the center point is found, the start, end and center points are passed to the curve algorithm. This algorithm in turn divides the arc into segments and calls the vehicle rendering method with the appropriate coordinates.
Acknowledgements

I would like to take a moment to acknowledge a few people without whom this thesis would not have been possible. First and foremost is my thesis advisor, Professor Ronald Mourant, thank you for all your invaluable inputs and guidance. It has been a great pleasure to have worked with you in the Virtual Environments Laboratory, Northeastern University. I will always remember your advice through the coming years and in my varied future interests.

Secondly I would like to thank my friend Anil K Potta in giving me the encouragement and to listen to my progress report every time he had called. I would also like to thank Amrita Mehta who has been phenomenal in giving inspiring pep talks and making me realize my potential. I also thank Zhishuai Yin for helping me out in the math part of my thesis. Thank you to all my other lab mates of the present and past Sonali Gupta, Na Chen, Linzhen Nie, Prasanna Rengarajan, and Namrata Gundewadi.

Finally I would like to thank my parents Narayanan and Mallika, my sister and brother-in-law and my lovely, little niece Nisha. I could not have done this without their love and support.
# Table of contents

<table>
<thead>
<tr>
<th>CHAPTER 1 INTRODUCTION</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 MOTIVATION</td>
<td>1</td>
</tr>
<tr>
<td>1.2 OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>1.3 WHAT IS A VIRTUAL ENVIRONMENT?</td>
<td>2</td>
</tr>
<tr>
<td>1.4 PRIOR WORK</td>
<td>3</td>
</tr>
<tr>
<td>1.5 WHAT CAN TO BE DONE TO MAKE VIRTUAL ENVIRONMENT REALISTIC?</td>
<td>4</td>
</tr>
<tr>
<td>CHAPTER 2 CURVE ALGORITHMS AND SEGMENTS</td>
<td>5</td>
</tr>
<tr>
<td>2.1 CURVES</td>
<td>5</td>
</tr>
<tr>
<td>2.2 ELEVATED CURVE GENERATION ALGORITHM</td>
<td>5</td>
</tr>
<tr>
<td>2.3 FLAT CURVE GENERATION ALGORITHM</td>
<td>15</td>
</tr>
<tr>
<td>2.4 WHY TWO ALGORITHMS?</td>
<td>25</td>
</tr>
<tr>
<td>CHAPTER 3 DATABASE MAINTENANCE OF ROADS AND AUTONOMOUS VEHICLES</td>
<td>26</td>
</tr>
<tr>
<td>3.1 THE VIRGINIA PROJECT</td>
<td>26</td>
</tr>
<tr>
<td>3.2 ROADS</td>
<td>26</td>
</tr>
<tr>
<td>3.2.1 How is the road text file formatted?</td>
<td>27</td>
</tr>
<tr>
<td>3.2.2 How is the road_data text file read and implemented by the algorithm?</td>
<td>29</td>
</tr>
<tr>
<td>3.2.3 Orientation change of Main Vehicle on hills</td>
<td>30</td>
</tr>
<tr>
<td>3.3 AUTONOMOUS VEHICLES</td>
<td>33</td>
</tr>
<tr>
<td>3.3.1 How is the autonomous vehicle’s text file formatted?</td>
<td>34</td>
</tr>
<tr>
<td>3.3.2 How is the auto_data text file read and implemented by the algorithm?</td>
<td>38</td>
</tr>
<tr>
<td>CHAPTER 4 MODIFYING THE ROAD DATABASE</td>
<td>41</td>
</tr>
<tr>
<td>4.1 STEPS</td>
<td>41</td>
</tr>
<tr>
<td>4.2 OTHER MODIFICATIONS TO THE ENVIRONMENT</td>
<td>47</td>
</tr>
<tr>
<td>4.3 LIMITATION OF IMPLEMENTING ALGORITHM IN THE DRIVING ENVIRONMENT</td>
<td>47</td>
</tr>
<tr>
<td>CHAPTER 5 MODIFYING THE AUTONOMOUS VEHICLE DATABASE</td>
<td>48</td>
</tr>
<tr>
<td>5.1 STEPS</td>
<td>48</td>
</tr>
<tr>
<td>CHAPTER 6 SUMMARY AND FUTURE WORK</td>
<td>55</td>
</tr>
<tr>
<td>6.1 ELEVATED CURVE</td>
<td>55</td>
</tr>
<tr>
<td>6.2 FLAT CURVE</td>
<td>56</td>
</tr>
<tr>
<td>6.3 FUTURE WORK AND IMPROVEMENTS</td>
<td>58</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>59</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>60</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>63</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1.1. Curve Creation – Before and After ................................................................. 1
Figure 1.2. Virtual Driving Environment ........................................................................ 3
Figure 2.1. Elevated Curve ............................................................................................... 5
Figure 2.2. Elevated Curve Models .................................................................................. 1
Figure 2.3. Curve Dimensions of the Flat-Concave Model of a Curve (SE) ..................... 7
Figure 2.4. Curve Dimensions of Convex-Flat Model of a Curve (SE) ............................. 8
Figure 2.5. Calculations of Increments .......................................................................... 11
Figure 2.6. Complete Road with Elevated-Curve Algorithm ......................................... 15
Figure 2.7. Models of Flat-Curve .................................................................................... 16
Figure 2.8. Flat Curve Angle Increment Calculation ..................................................... 18
Figure 2.9. Calculation of New Coordinates ................................................................... 20
Figure 3.1. road_data.txt ............................................................................................... 27
Figure 3.2. Calculation of Orientation Angle of Driver Vehicle and Its Mirrors .......... 31
Figure 3.3. auto_data.txt ............................................................................................... 35
Figure 4.1. Elevated Curve (SE) Radius Calculations ..................................................... 44
Figure 5.1. Discrepancies with Two Contiguous Curves ............................................... 49
Figure 5.2. Discrepancy Solved by Having a Line Segment ........................................... 49
Figure 5.3. Approximate Arc .......................................................................................... 50
Figure 5.4. The Fillet Command Inserts a Tangent Arc and Trims the Lines Accordingly .............................................................................................................. 51
Figure 5.5. Curve After Fillet Operation .......................................................................... 52
Figure 5.6. Curve Comparison ....................................................................................... 53
Chapter 1 Introduction

1.1 Motivation

The Virtual Environments Laboratory at Northeastern University has built a 3D virtual environment based on real roads in Charlottesville, Virginia. The environment was designed by taking videos of roads and pictures of buildings, signs, and traffic signals in the Charlottesville area. The videos were used for the roads and the pictures were used for the textures of roads, buildings, signs etc. We built into this environment various scenarios to equate virtual reality to reality. A few examples of scenarios are, opposite traffic driving across the middle lane marker, pedestrians crossing the road, traffic lights etc. This environment can be used as a driving evaluation to find out the point in a medical condition when a person should stop driving in the real world. The basic design of the environment has been completed (as of September ’08) but the roads and autonomous vehicles have not yet been face validated.

1.2 Overview

This thesis is concerned with the procedure for making the un-even roads smooth, and autonomous vehicles travel smoothly on curved and hilly roads; thereby face-validating the environment. In the present design of the environment a car goes on a curved road taking sets of straight paths. My project changes the path of autonomous vehicles to curves.
This is the same when rendering elevated roads. The roads were sets of straight paths and I have changed them to make it smooth paths. Though curves are fundamentally sets of straight lines, if the number of segments (of straight lines) is very high then the road will look like perfect curves but will take a lot of processing power. It is the designer’s responsibility to find an optimum balance and pass the number of segments as input to the algorithm.

1.3 What is a Virtual Environment?

According to an IEEE paper (Ellis, 1994), Virtual Environment displays are interactive, head-referenced computer displays that give users the illusion of displacement to another location. Thus, we can define virtual environments as interactive, virtual image displays enhanced by special processing and by non-visual display modalities, such as auditory and haptic, to convince users that they are immersed in a synthetic space. A major challenge of a virtual environment is to convince users to react to synthetically-generated scenarios as if they were real. Another major challenge is to use the limited the resources at hand to manage the virtual environment. This is why the efficiency of algorithms used to render the environment is of utmost importance.

Virtual environments are used in many fields today to train, evaluate, and study human performance (van Dam, Forsberg, LaidLaw, LaViola, Simpson, 2000). The wide variety of fields that make use of virtual environments encompass Medical, Teaching, Military, Psychology, Entertainment etc.

Virtual reality systems consist of one or more computers, a few projectors, and a screen on which the environment is projected. An example of a typical driving environment is displayed in Figure 1.2.
Figure 1.2. Virtual driving environment.

Driving simulators have provided a safe environment to measure driver performance with repeatability among subjects. Several studies (Plumert, Kearney, Cremer, 2004) have focused on validating virtual environments’ capabilities to adequately simulate the real world. There is a large variety of simulators - from high fidelity motion based simulators and cave-in type simulators to a desktop computer with a joystick. The advantage of simulators over real life testing is their capability to present a subject with real-life situations that can affect humans or destroy something including the driver.

1.4 Prior Work

Road design in virtual environments is not a well-documented area of research. Simulation software and game developers have an interest in maintaining proprietary systems to minimize competition. Much work remains in creating an open design format.
Transportation engineers use visualization tools for highway design. Their emphasis however, is on creating geometry that models a regulated standard. The attention of this thesis however, is on modifying the road and autonomous vehicle database in *driving simulations* for face-validation. While previous research (Oza, 2006) has dealt with rendering curved roads using splines, this thesis concentrates on an easier and less complex road rendering algorithm.

### 1.5 What can to be done to make Virtual Environment realistic?

One of the astonishing and distinguishing features of virtual interface technology is a participant's response to it as a place. While experiencing virtual worlds, they can be heard saying "I wonder what's in *here*" or, "*where* I am now?", and afterwards, their comments about their experience often start with the words "when I was *there*". "*Here, there and where*" are responses to a place. Some people who experience virtual worlds feel as if they are really there, in the virtual models. Few other presentation mediums, such as cinematography, television, books, music, or architectural renderings, can generate such compelling impressions of being in a place.

There are a few fundamental criteria to virtual reality which are responsible for creating the effect of being in a realistic world. In virtual driving environments it is the roads being smooth curves and not a set of straight segments. The other one would be the movement of autonomous vehicles. The vehicles should not always be going in straight paths even if the road is curved.
Chapter 2 Curve Algorithms and Segments

2.1 Curves

Since vehicle interaction is integral to the driving experience, it is important that the simulated traffic appear to behave and interact realistically with the subject driver in the simulation environment (Al-Shihabi and Mourant, 2003). Curves are an most important aspect in making a virtual environment realistic. This chapter deals with the various algorithms used in the thesis for elevated roads and curves for autonomous vehicles. Curves are essentially sets of numerous straight lines so closely placed that the illusion of a curve is produced. A typical curve generation takes in the start, end and center points as input and give out numerous segments (four corner points for each segment) that are placed contiguously.

2.2 Elevated curve generation algorithm

This algorithm is devised by taking into consideration that the start point of the curve subtends a $90^\circ$ angle with the $y$-axis as illustrated in Figure 2.1.

![Figure 2.1. Elevated curve.](image)
We also take into consideration that the centre of the circle, of which the arc is a part, is on the Y-axis. To summarize, the line joining the center of the circle and the start point of the arc are parallel to the Y-axis.

There are four possible models using this algorithm.

1) Flat-Concave
2) Concave-Flat
3) Flat-Convex
4) Convex-Flat

The following diagrams (Figure 2.3 and 2.4) are references for the explanation of the algorithm.
Figure 2.3. Curve dimensions of the flat-concave model of a curve (SE).
Figure 2.4. Curve dimensions of convex-flat model of a curve (SE).
Green arc – The wanted curve
S – Starting point of arc
E – End point of arc
C – Center of circle formed by arc.
R – Radius of circle
A – Angle to be found
YDIFF – Difference in the y-values of the start and end points of the arc

Figures 2.3 and 2.4 illustrate two of the models, Flat-Concave and Convex-Flat respectively.

**Input**: Start point, End point, Radius, Model to be rendered, Width of road, Total number of segments, Width of road, Texture image, First Segment to be rendered, Last segment to be rendered.

**Output**: Four corners of each road segment.

**Steps**:

1) Find Angle A from $\theta_1$ and $\theta_2$.

2) Find angle increment

3) For each angle increment,

   a. Calculate increments in x, y and z axes.

   b. Pass the present coordinates (x, y, z) and new coordinates(x ± xincr, y ± yincr, z ± zincr) to another function that computes the four corners of the segment.

**Step 1 – Find Angle A**
\[ A = \theta_2 - \theta_1 \]

WHERE,
\[ \theta_1 = \frac{3\pi}{2} \]
\[ \theta_2 = 2\pi - \arcsin\left(\frac{R - Y\text{DIFF}}{R}\right) \]
\[ = 2\pi - B \]

AND
\[ \theta_2 = \frac{3\pi}{2} \]
\[ \theta_1 = \pi + \arcsin\left(\frac{-Y\text{DIFF} + R}{R}\right) \]
\[ = \pi + B \]

Therefore,
\[ A = \frac{\pi}{2} - B \]

**Step 2 – Find Angle Increment (\( \delta \)) and Angle subtended by curve on XZ-plane (\( \alpha \))**

\[ \delta = \frac{A}{\text{numSegments}} \]

WHERE, numSegments = Number of Segments

\[ \alpha = \arctan\left(\frac{pe.z - ps.z}{pe.x - ps.x}\right) \]
Step 3 – Iterate through segments

Step 3.1 – Calculate increments of coordinates

Figure 2.5. Calculations of increments.
\( \delta \) = Angle increment
\( R \) = Radius of circle formed by arc SE
\( \text{YINCR} \) = Increment along the y-axis
\( \text{XZINCR} \) = Increment along the XZ plane

Calculations (illustrated in Figure 2.5)

\[
\text{YINCR} = \text{CY}_1 - \text{CY}_2
\]
\[
\text{CY}_1 = R \cdot \cos(\delta)
\]
\[
\text{CY}_2 = R \cdot \cos(2\delta)
\]

\[
\text{XZINCR} = \text{M}_2 \text{Y}_2 - \text{M}_1 \text{Y}_1
\]
\[
\text{M}_2 \text{Y}_2 = R \cdot \sin(2\delta)
\]
\[
\text{M}_1 \text{Y}_1 = R \cdot \sin(\delta)
\]

\[
\text{XINCR} = \text{XZINCR} \cdot \cos(\alpha)
\]
\[
\text{ZINCR} = \text{XZINCR} \cdot \sin(\alpha)
\]

Step 3.2 – Calculate new coordinates

\[
\text{NEWX} = \text{OLDX} \pm \text{XINCR}
\]
\[
\text{NEWY} = \text{OLDY} \pm \text{YINCR}
\]
\[
\text{NEWZ} = \text{OLDZ} - \text{ZINCR}
\]

Step 3.3 – Pass coordinates to another function
The old and the new coordinates are passed on to another function that determines the four corners of the road.

Below is the source providing the core functionality of the algorithm.

```cpp
void Roads::populateElevationCurvedRoad(MyMap& myMap, const Point3f& ps, const Point3f& pe, float width, float radius, const char* fileName, int segments, int firstSegment, int finalSegment, int model, float ydiff, string segname, const char* segname1)
{
    int xdir=0, seg=1;
    float yIncr = 0.0f;
    float xIncr = 0.0f;
    float zIncr = 0.0f;
    float xzIncr = 0.0f;

    float x = ps.x;
    float y = ps.y;
    float z = ps.z;

    Point3f prev = ps;

    if(pe.x > ps.x)
        xdir = 1;
    else
        xdir = -1;

    float theta1 = (float)(M_PI/2);
    float theta2 = (float)(2*M_PI+asin((ydiff+radius*sin(theta1))/radius));

    switch(model){
        case 1://FLATCONCAVE:
            theta1 = 3*theta1;
            theta2 = (float)(2*M_PI+asin((ydiff+radius*sin(theta1))/radius));
            break;
        case 2://CONCAVEFLAT:
            ydiff -= (2*ydiff);
            theta2 = 3*theta1;
            theta1 = (float)(M_PI+asin((ydiff-radius*sin(theta2))/radius));
            break;
        case 3://FLATCONVEX:
            theta1 = 3*theta1;
            theta2 = (float)(2*M_PI+asin((ydiff+radius*sin(theta1))/radius));
            break;
    }
}
```
break;
case 4://CONVEXFLAT:
ydiff -= (2*ydiff);
theta2 = 3* theta1;
theta1 = (float)(M_PI+asin((ydiff-radius*sin(theta2))/radius));
break;
default: printf("Please provide a model between 1 and 4");
}

float stepAng = (theta2 - theta1)/segments;
float alpha = (float)atan( (pe.z-ps.z)/(pe.x-ps.x) );
int upOrDown;

int i = 1; //unique string append
char unique_append[3];

while( theta2 > (theta1+0.0005f)){
    yIncr  = radius * (float)( sin(theta1 + stepAng) - sin(theta1) )
    xzIncr = radius * (float)( cos(theta1 + stepAng) - cos(theta1) )
    xIncr  = xzIncr * (float)cos(alpha);
    zIncr  = xzIncr * (float)sin(alpha);
    theta1 += stepAng;
    
    if((model == 1) || (model==2))
        y += yIncr;
    else
        y -= yIncr;
    
    if(xdir==1)
        x += xIncr;
    else
        x -= xIncr;
    
    z -= zIncr;

    Point3f endPoint;
    endPoint.x = x;
    endPoint.y = y;
    endPoint.z = z;
    itoa(i+1,unique_append,10);
    segname = segname.append(unique_append);
    if((seg <= finalSegment) && (seg >= firstSegment)){
        if(pe.z > ps.z)
            populateRoadSeg(myMap,endPoint, prev, width, fileName, 1, segname,

14
Using the above algorithm, a complete road between two roads having a consistent y-coordinate can be formed by interspersing the four models in the order Convex-Flat, Flat-Concave, Concave-Flat, Flat-Convex. Figure 2.6 illustrates this.

![Figure 2.6. Complete road with elevated-curve algorithm.](image)

**2.3 Flat Curve generation algorithm**

This algorithm does not consider the $90^0$ rule as is the case of the previous algorithm. Thus, there are 2 possible curves using this algorithm

1) Clockwise
2) Counter-clockwise

![Figure 2.7. Models of flat-curve.](image)

S – Start Point  
E – End Point

**Input:** start point, end point, velocity, center of circle formed by curve, string indicating clockwise or counter-clockwise motion.

**Output:** Next point on curve for \( \delta \) time. This point is then passed on to render the vehicle.

**Steps:**

1) Initialize the first point of algorithm to the start point of the curve.

2) For each increment till the end point is reached, do the following

   a) Calculate angular increment \( (\alpha) \)

   b) Get new values for the coordinates \((x, y, z)\)

      i) Get initial displacement angle \( (\beta) \) between Center C and Start point S.

      ii) Calculate displacement for new position after \( \alpha \) increment from the Center C along the X and Z axes
iii) Calculate displacement for new position after $\alpha$ increment from the Start point along the Y-axis

iv) Calculate rotation angle for the vehicle along all the axes

v) Define new coordinates

c) Pass the new coordinates to the vehicle render function

**Step 1 Initialize first point**

Pass first point of the curve to the car render method. The car is then rendered at the start point of the arc.

**Step 2 Iterate till end point of arc is reached**

**Step 2.1 Calculate angular increment ($\alpha$)**
Figure 2.8. Flat curve angle increment calculation.

S – Starting point of arc SE (point E not shown in Figure 2.8)

C – Center of arc SE

S1 – Point after first iteration

L – Displacement for first iteration

R – Radius of circle formed by arc SE

\( \alpha \) – The angle increment to be found
Calculations (illustrated with help of Figure 2.8)

\[
\sin\left(\frac{\alpha}{2}\right) = \frac{L/2}{R}
\]

\[
\left(\frac{\alpha}{2}\right) = \arcsin\left(\frac{L/2}{R}\right)
\]

Since, L is very small for 1 millisecond,

\[
\arcsin\left(\frac{L/2}{R}\right) = \left(\frac{L}{2R}\right)
\]

\[
\left(\frac{\alpha}{2}\right) = \left(\frac{L}{2R}\right)
\]

\[
\alpha = \frac{L}{R}
\]

Therefore, Angle Increment (\(\alpha\)) = \(\frac{L}{R}\)

WHERE,

\[
L = (\text{velocity} \times \delta t) + \frac{1}{2} (\text{acceleration} \times (\delta t)^2)
\]

R is given as input in the database

Step 2.2 Get new coordinates
Figure 2.9. Calculation of new coordinates.

SE – The path to be taken by the autonomous vehicle

$S_1$ – New position after first iteration

$\alpha$ - Step angle increment
\( \beta \) - Initial slope of line segment SC

R – Radius of circle formed by arc

\( \delta x \) – Displacement of new coordinate \( S_1 \) from the center C along X-axis

\( \delta y \) – Displacement of new coordinate \( S_1 \) from the center C along Y-axis

**Step 2.2.1 Get initial angle (\( \beta \)) between Center (C) and Start of arc (S)**

\[
\beta = \text{atan} \left( \frac{SP_2}{CP_2} \right)
\]

WHERE,

\( SP_2 = S.z - C.z \)

\( CP_2 = S.x - C.x \)

**Step 2.2.2 Calculate displacement from Center after \( \alpha \) angle increment along X and Z coordinates**

\[
S_{1,x} = C.x + \delta x
\]

\[
S_{1,z} = C.z + \delta z
\]

WHERE

\[
\delta x = R \times \sin(\beta + \alpha)
\]

Since, \( \sin(\beta + \alpha) = \delta x / R \)

\[
\delta z = R \times \cos(\beta + \alpha)
\]

Since, \( \cos(\beta + \alpha) = \delta z / R \)
Step 2.2.3 Calculate increment along y-axis

We calculate the increment along y-axis because this algorithm takes into consideration a linear increase in elevation (y).

\[ \delta y = \frac{YDIFF}{numSegments} \]

WHERE,

\( YDIFF \) – Difference in Y values of Start and End points
\( numSegments \) – Number of segments

Calculation of Number of Segments

\[ numSegments = \frac{arc Length}{Displacement} \]

WHERE

\( arc length = \text{radius} \times \text{(total angle subtended by arc at center)} \)
\( Displacement = L \) (length of line segment SS\(_1\))

Therefore,

\[ \delta y = \frac{YDIFF \times Displacement}{arc length} \]

Step 2.2.4 Calculate Rotation angles

Convert all angles from degrees to radians and pass them as parameters. The angular increment along the z-axis should be smooth and thus it needs more calculations.

New angle about X-axis (NewXAngle) = (X-angle input) \( \times \frac{180}{\pi} \)

New Angle about Y-axis (NewYAngle) = (Y-angle input) \( \times \frac{180}{\pi} \)

New Angle about Z-axis (NewZAngle) = Z_angle
WHERE,

$$Z\text{-angle} = \text{Input Z-angle} + (-\beta \ast \left(\frac{180}{\pi}\right))$$

If, curve is clockwise

$$Z\text{-angle} = \text{Input Z-angle} + (-\beta \ast \left(\frac{180}{\pi}\right)) - 180^0$$

If, curve is anti-clockwise

**Step 2.2.5 Define new coordinates**

New X position = prevX + \(\delta x\)

New Y position = prevY + \(\delta y\)

New Z position = prevY + \(\delta z\)

New X-axis angle = NewXAngle

New Y-axis angle = NewYAngle

New Z-axis angle = NewZAngle

**Step 2.3 Pass parameters to render method**

The render method is in another class named *Ms3DCarFactory* and is named *Ms3DCarFactory_render()*.

All the new coordinates and new angles are passed to this function along with the vehicle’s model class. The definition of this method is,

```c
void Ms3DCarFactory_render(AutonomousVehicle autoVehicleModel, float x, float y, float z, float x_angle, float y_angle, float z_angle, int path_no);
```

Below is the source providing the core functionality of the algorithm.

pc – Coordinates of center point of circle formed by curve

ps – Coordinates of start point of curve
pe – Coordinates of end point of curve
autoVehicleModel – Autonomous vehicles model class
Initial ps_ang – β
Initial z_angle – α

```cpp
void AutonomousMove::getNextCurvePoint()
{
    float radius = (float) (sqrt( (pc[0] - ps[0]) * (pc[0] - ps[0]) + (pc[1] - ps[1]) * (pc[1] - ps[1])));

    float displacement = (((initial_vel*deltatime)/1000) +
                          (veh_acc*deltatime*deltatime) / (2*1000000));
    float initial_vel = initial_vel+((veh_acc*deltatime)/1000);

    float dTotal = (float) ( radius * theta ); //the total arc length
    float deltaY = ((pe[2] - ps[2]));

    ps_pc[0] = ps[0]-pc[0];
    ps_pc[1] = ps[1]-pc[1];
    float ps_pc_slope = ps_pc[1]/ps_pc[0];
    float ps_ang = atan(ps_pc_slope);

    static float z_angle=0;
    zAngle = displacement/radius;
    if(!curve_clockwise)
        zAngle = -zAngle;
    ps_ang += zAngle;
    z_angle += zAngle;

    x_value = pc[0] + radius * (float) (cos(ps_ang));
    y_value += deltaY* displacement/dTotal;
    z_value = pc[1] + radius * (float) (sin(ps_ang));

    x_angle = xAngle*180/M_PI;
    if(!curve_clockwise)
        z_angle = (-ps_ang*57.3)-180.0+autoVehicleModel->rotate_z;
    else
        z_angle = (-ps_ang*57.3)+autoVehicleModel->rotate_z;
    y_angle = yAngle*180/M_PI;

    if (!culled) {
        // Code continues here...
    }
}
```
2.4 Why two algorithms?

There are two algorithms because either of the algorithms cannot draw the other curve.

In the case of Algorithm 1 (Elevated Curve Algorithm) there is the constraint that the curve always starts perpendicular or parallel to the y-axis. I safely assumed this because elevated roads are always connected to flat (constant y) roads. In case of flat-curved roads, the road is seldom parallel or perpendicular to an axis. Thus, Algorithm 1 cannot be used to create flat-curved roads.

In the case of Algorithm 2 (Flat Curved Path Algorithm) there is no constraint. Thus, Algorithm 2 can be used to create elevated curves too but, to avoid any gaping holes in the curve; we need to overlap adjacent curves. Overlapping adjacent sections of the curve is not advisable because it is not feasible to make the texture continuous. Thus, Algorithm 2 cannot be used to render elevated-curved roads.
Chapter 3 Database Maintenance of Roads and Autonomous Vehicles

3.1 The Virginia Project

I have implemented the aforementioned algorithms in the Virginia Project. This project has multiple scenarios namely, Scenario_1A, Scenario_1B, Scenario_2A, and Scenario_2B. The specifications for the scenarios were provided to us mainly by Dr. Daniel Cox at the University of Virginia at Charlottesville. Each scenario is divided into tiles so as to reduce system load by dynamically loading tiles in use and unloading tiles already used. There are 14 tiles in each scenario. Each of the scenarios has multiple data files. A few examples of this are sign data, autonomous vehicle data, traffic-light data, trigger data etc. The autonomous vehicles, signs, etc. are different in each scenario and thus are scenario-specific. There are also many data files that are common to all scenarios. A few examples of this are road data, tree data, building data etc.

The following sections will explain the road data files (part of the common data files) and the autonomous vehicle data files (part of the scenario-specific data files). The road data file is used to implement Algorithm 1 (Elevated Curve Algorithm) and the autonomous vehicle data file is used to implement Algorithm 2 (Flat Curve Algorithm).

3.2 Roads

This part of the thesis deals with elevated-curved roads. This section will deal with how the database for roads is maintained, read and implemented in the algorithm.
3.2.1 How is the road text file formatted?

Part of a typical road data file is illustrated in Figure 3.1.

![Figure 3.1. road_data.txt.](image)

Each line specifies a road segment. Within a line, each value is separated by a space or tab. Each line should either start with #, 0, 1, 3, 4, 5, 6, 9 or be an empty line. The first character of a line determines what is to be rendered.
# - Comment line

0 – Straight flat road
1 – Curved flat road
3 – Straight flat grass
4 – Curved flat grass
5 – Elevated curve (The one we are interested in.)
6 – Culled straight grass
9 – Irregular road

A line starting with the number 5 is what is used for creating elevated curved roads. There should be 16 variables in a line pertaining to elevated curved roads. The following are the 16 variables and a brief explanation of each.

1) Road ID – 5
2) Start point’s X coordinate – *should be a floating point number*
3) Start point’s Y coordinate – *should be a floating point number*
4) Start point’s Z coordinate – *should be a floating point number*
5) End point’s X coordinate – *should be a floating point number*
6) End Point’s Y coordinate – *should be a floating point number*
7) End Point’s Z coordinate – *should be a floating point number*
8) Width of the road – *should be a floating point number*
9) Radius of the circle formed by the curve – *should be a floating point number*
10) Texture image file – *should be the name of an image file in the images folder of the project*
11) Number of segments – *should be a whole number*
12) The first segment to be rendered – *should be a whole number less than or equal to number of segments*

13) The last segment to be rendered – *should be a whole number less than or equal to number of segments and greater than first segment*

14) The model of the road. It should be

   1, if Flat-Concave
   2, if Concave-Flat
   3, if Flat-Convex
   4, if Convex-Flat

15) The difference between the start point’s y-coordinate and end-point’s y-coordinate. *It should be absolute value of (startY – endY).*

16) A unique name for this segment of the road - *alphanumeric*

   If any one of the variables is not specified properly, then the whole road segment is not rendered.

### 3.2.2 How is the road_data text file read and implemented by the algorithm?

There are various scenario languages to implement driving simulation (Kearney and Willemsen, 1999). Visual C++ was used to implement the virtual environment using OpenGL and SDL Libraries. The C++ file that was used to read and implement the Elevated Curve Algorithm is *Roads.cpp*.

The code segment that reads the elevation curved road line is given below.

```cpp
if ('5'==line[0])
{
    retCode = sscanf(line, "%d%f%f%f%f%f%f%f%f%s%d%d%d%d%f%s", &isCurve,
                    &startX, &startY, &startZ, &endX, &endY, &endZ, &width, &radius,
```
(fileName, &segments, &firstSegment, &finalSegment, &model, &ydiff, segname);
if (retCode != 16) {
    fprintf(stderr, "%s:%d: error: pattern mismatch: %s\n",
            DRIVINGSIM ROAD_DATA_FILE, lineCount, line);
    continue;
}
pe.set(startX, startY, startZ), ps.set(endX, endY, endZ);
populateElevationCurvedRoad(myMap, ps, pe, width, radius, fileName, segments, firstSegment, finalSegment, model, ydiff, segname, segname);
roadCount++;
}

As the above code segment illustrates, the C function `scanf()` is used to scan each line. The first character of the line is checked initially to determine the variables to expect. As noted earlier, for an elevated curved road, the first character should be 5. Variable `retcode` is used to determine the number of parameters scanned. If it is not equal to 16, this line of the data file is ignored and the road is not rendered, else, the algorithm function is called with all the parameters passed to it.

The function `populateElevationCurvedRoad()` determines if the data types of each of the 16 variable is valid. Type-casting is used to determine the same. Example, `float x = (float)ps.x`. This statement ensures that `ps.x` is a floating point number. A runtime exception is thrown otherwise.

### 3.2.3 Orientation change of Main Vehicle on hills

Another important aspect of face-validation is to orient the main vehicle (rather, the view of the 3D scene) according to the slope of a curve. Non-existence of this would make the driver of the simulator feel like floating on elevations. This also encompasses the aspect of rotating the mirrors (left side-view mirror and the rear-view mirror) in the other direction.
In order to achieve the above rotation, the angle difference between consecutive points should be calculated and passed on to the MainVehicleDriver’s render function.

Consider Figure 3.2 for the explanation of the calculation of angle.

![Figure 3.2. Calculation of orientation angle of driver vehicle and its mirrors.](image)

Consider two consecutive points in the vehicle’s path A and B while the driver is going up a hill. The orientation of the main vehicle should change by the slope of the curve. The slope angle of the curve in Figure 3.2 is $\alpha$.

$$\sin(\alpha) = \frac{\text{YDIFF}}{L}$$

Therefore,

$$\alpha = \arcsin\left(\frac{\text{YDIFF}}{L}\right)$$

where,

$L$ – Straight line distance between 2 consecutive points in the vehicle’s path
YDIFF – Difference in y-coordinates of the above points

This angle has to be passed on to the function that updates the coordinates of the driver according to the control mechanism (mouse, joystick, external/internal dll). The OpenGL command `glRotatef()` is used to determine the new orientation. The rotation should be along the x-axis for the desired effect, and thus the code segment that effects the angle change is,

```
glRotatef(\alpha ,1.0f,0.0f,0.0f).
```

Though the angle of rotation is the same, the direction of rotation for the mirrors (side-view and rear-view) is the opposite of the main vehicle. Thus the rotate function is,

```
glRotatef(-\alpha ,1.0f,0.0f,0.0f).
```

The angle of rotation (\( \alpha \)) being a significant angle, passing it to the rotate function is not advisable. The immediate, large angle change would make the ride very jerky. To make the angle change smooth, \( \alpha \) was divided into a number of segments. For each iteration of the updated motion method, a segment of the angle is passed as argument to the `glRotatef()` function. The number of segments is so small that the delay is acceptable and the transition/rotation is also smooth.

The code segment that corresponds to the above explanation is,

```c
float ydiff = yprevi - currentPos.y;
theta1 = asin(ydiff/displacement_value)*180/M_PI;
```

```c
if(displacement_value > 0)
{
    if(numTimes >= numSegs) {
        flag = false;
    }
    ...
}
```

```c
theta1 = asin(ydiff/displacement_value)*180/M_PI;
```
numTimes = 0;
}
if(!flag) {
    theta2 = 3*theta1/numSegs;
    flag = true;
}
numTimes++;
}

if(displacement_value > 0)
{
    if(rotateAngle == 0)
        glRotatef(theta2, 1.0f, 0.0f, 0.0f);  //Main View
    else
        glRotatef(-2*theta2, 1.0f, 0.0f, 0.0f); // Mirrors
}

numSegs – The number of segments. For a smooth but fast rotation should be between 15 and 20.

displacement_value – Corresponds to $L$ in Figure 3.2. It is found using the formula for distance between 2 points i.e., $((\text{difference in x})^2 + (\text{difference in y})^2 + (\text{difference in z})^2)$

theta1 – The total angle to be rotated.

theta2 – The segment angle.

rotateAngle – The angle of rotation of the scene for mirrors

### 3.3 Autonomous Vehicles

Another part of this dissertation deals with curved paths of autonomous vehicles. This section deals with the maintenance and implementation of the database for autonomous vehicles.
As noted previously, vehicle data is a scenario specific data file. Thus, there are numerous autonomous vehicle database text files in each scenario. To be more specific, there is one text file for autonomous vehicles in each tile. To sum it all up, there are 14 autonomous vehicle database files in each scenario and since there are 4 scenarios, there are 56 autonomous vehicles database files. The database text file is named `auto_data.txt`.

### 3.3.1 How is the autonomous vehicle’s text file formatted?

Part of a typical `auto_data.txt` file is illustrated in Figure 3.2
Figure 3.3. auto_data.txt.

Each path segment is specified by 22 parameters each separated by a comma. To have a better understanding of the data, there is a line break after every 22 parameters and thus each line specifies a path segment.

Each set of 22 parameters should start with the first character being *, $, space character or first character of a vehicle model. This character determines the action of the application.

* - Comment Line
$ - Specifies that the vehicle’s velocity is related to the speed of the driver vehicle. This is used to implement any scenario that would require an autonomous vehicle to be at a position relative to the driver vehicle.

Apart from the above mentioned characters, there is a special case of the autonomous vehicle’s wait information line. A wait line typically has 6 parameters. A wait line is differentiated from the other lines of data by starting the line with # character followed by vehicle’s model name.

Unless the line is empty, the first character is followed by a vehicle’s model name. The 22 parameters (separated by commas) for a typical non-waiting, non-comment line of data is as follows,

1) Vehicle Model Name – This is in some cases preceded by $ to indicate the autonomous vehicle’s velocity’s relation with the driver vehicle’s velocity – should be a string with a model name available in the models folder of the project

2) Vehicle ID – Every vehicle has a unique ID – should be alphanumeric

3) Path type – can be curvePath or straightPath. As the name suggests, curvePath specifies that the path segment is curved and straightPath specifies that the path segment is straight.

4) Velocity – This will specify the velocity of the autonomous vehicle. If the first character of the first parameter is $, then this variable is a factor e.g., 0.5, 1.0 etc. 0.5 specifies that the autonomous vehicle in consideration goes at half the speed of the driver vehicle. 1.0 specifies that the vehicle goes at exactly the same speed as the driver vehicle. If the first character is different from $, then this value specifies the absolute velocity of the vehicle in consideration – should be floating point number
5) Acceleration – This specifies the acceleration of the autonomous vehicle – *should be a floating point number*

6) Deceleration – This variable has the effect opposite to acceleration – *should be a floating point number*

7) Start point’s X coordinate – *should be a floating point number*

8) Start point’s Y coordinate – *should be a floating point number*

9) Start point’s Z coordinate – *should be a floating point number*

10) End point’s X coordinate – *should be a floating point number*

11) End point’s Y coordinate – *should be a floating point number*

12) End point’s Z coordinate – *should be a floating point number*

13) Turn variable.

   a) *clockwise* or *anticlockwise*, if the path type is *curvePath*

   b) *none*, if path type is *straightPath*

The following 3 variables are considered by the algorithm only if the path type is *curvePath*.

Consider that $C$ is the center of the circle formed by the curve.

14) $C$’s X coordinate – *should be a floating point number*

15) $C$’s Y coordinate – *should be a floating point number*

16) $C$’s Z coordinate – *should be a floating point number*

17) Scale Factor – This specifies the scale factor of the vehicle’s model – *should be a floating point number*

18) Angle of rotation about the X-axis – *should be a floating point number*

19) Angle of rotation about the Y-axis – *should be a floating point number*

20) Angle of rotation about the Z-axis – *should be a floating point number*

21) Indicator render variable. This is model specific. For a black benz model,
a) 0 – for no indicators or turn signal  
b) 1 – for left turn indicator  
c) 2 – for brake light  
d) 3 – for right turn indicator

22) Traffic-light Id – This is used to make the autonomous vehicle follow the trigger for traffic-lights i.e., car proceeds only if the traffic-lights are green – should be a whole number

Similar to road database, any mistake in specifying the parameters will stop the autonomous car from being rendered.

### 3.3.2 How is the auto_data text file read and implemented by the algorithm?

A class named `AutonomousVehicleLoader` is used to load the data and convert it into application ready variables and classes. The autonomous vehicles data is read line by line and for each line variables are separated by comma. This is a character level read.

The code segment that does the read operation is given below.

```c
if ((line[i]! = ',')&&(var_no==1)&&(line[i] != ' ')&&(line[i] != '$')&&(line[i] != '@')&&(line[i] != '&'))
{
    *(vehicleName_dummy+char_loc)=line[i];
    if (line[i+1] == ',') *(vehicleName_dummy+char_loc+1)='\0';
    char_loc++;
}
if ((line[i]! = ',')&&(var_no==2)&&(line[i] != ' '))
{
    *(vehicleId_dummy+char_loc)=line[i];
    if (line[i+1] == ',') *(vehicleId_dummy+char_loc+1)='\0';
    char_loc++;
}
if ((line[i]! = ',')&&(var_no==3)&&(line[i] != ' '))
{
    *(type_dummy+char_loc)=line[i];
    if (line[i+1] == ',') *(type_dummy+char_loc+1)='\0';

```
char_loc++;
}
if ((line[i]!='')&&(var_no==4)&&(line[i]!=' '))
{
    *(vel_dummy+char_loc)=line[i];
    if (line[i+1] == ',') *(vel_dummy+char_loc+1)='\0';
    char_loc++;
}
if ((line[i]!='')&&(var_no==5)&&(line[i]!=' '))
{
    *(acc_dummy+char_loc)=line[i];
    if (line[i+1] == ',') *(acc_dummy+char_loc+1)='\0';
    char_loc++;
}
if ((line[i]!='')&&(var_no==6)&&(line[i]!=' '))
{
    *(deacc_dummy+char_loc)=line[i];
    if (line[i+1] == ',') *(deacc_dummy+char_loc+1)='\0';
    char_loc++;
}

And so on till all the 22 variables are read. There is another read section for vehicle wait information.

It can be seen that variables are loaded character-by-character into a few variables prefixed with the keyword dummy. To convert the sets of characters into application readable format, they are formed as variables. These variables are then loaded into an object of AutonomousVehicle class i.e, an autoVehicle object is created using the above mentioned variables.

strcpy(autoVehicle(autoVehicleNo).vehicleName,vehicleName_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).vehicleId = atoi(vehicleId_dummy);
strcpy((autoVehicle(autoVehicleNo).pathDescriptor(no_path).type),type_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).velocity = atof(vel_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).acceleration = atof(acc_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).deacceleration = atof(deacc_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).startpos_x = atof(start_x_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).startpos_y = atof(start_y_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).startpos_z = atof(start_z_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).endpos_x = atof(end_x_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).endpos_y = atof(end_y_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).endpos_z = atof(end_z_dummy);
strcpy(autoVehicle(autoVehicleNo).pathDescriptor(no_path).turn,turn_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).centerpos_x = atof(center_x_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).centerpos_y = atof(center_y_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).centerpos_z = atof(center_z_dummy);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).isindicator = atoi(is_indicator);
autoVehicle(autoVehicleNo).pathDescriptor(no_path).traffic_light_id = atoi(traffic_light_id);

atof(), atoi() and strcpy() are used to check for data type integrity. Any discrepancy in the data type will void the car from being rendered.
Chapter 4 Modifying the Road Database

The algorithm is as good as the input. The algorithm will create uneven curves or may even not render curves, if the center or radius of the circle formed by the arc is not appropriate. Thus, modifying the road database involves getting the right coordinates and the correct radius. I say modifying instead of creating because this thesis is concerned with smoothening already existing roads.

4.1 Steps

There are two steps to modifying the road database.

1) Determine start point and end point coordinates.

2) Determine the appropriate radius of the curve between the start and the end points.

Step 1 Find start and end coordinates

Determining the coordinates of the road in consideration is a manual process. One should drive through the environment to determine the start and end points of the road to be modified. Coordinates of any point in the driving environment can be determined by driving to that point and pressing a hotkey that outputs the coordinates of the same. It is important that the coordinates are taken by not turning the vehicle so as to make the textures of adjacent roads match.

The hot key in this thesis is F2. The source code that handles the F2 key press event is as follows,
void Main_handleKeyPressEvent(SDL_keysym* keysym, SDL_KeyboardEvent key)
{
    case SDLK_F2:
        if(key.type == SDL_KEYDOWN)
        {
            fprintf(stderr, "The region name is %s\n", strReg.data());
            fprintf(stderr, "Current driver location meter: %.2f, 
                MainVehicleDriver_location().x);
            fprintf(stderr, " %.2f,MainVehicleDriver_location().y");
            fprintf(stderr, " %.2f\n",MainVehicleDriver_location().z);

        }
        break;
}

The above method uses fprintf() to print the coordinates to standard error console (stderr). The
MainVehicleDriver_location() returns a Point3f object. The data structure of Point3f object is
given below.

class Point3f
{
    public:
        float x, y, z;
        inline Point3f() : x(0.0f), y(0.0f), z(0.0f) {}
        inline Point3f(const Point3f& p) : x(p.x), y(p.y), z(p.z) {}
        inline Point3f(float x, float y, float z) : x(x), y(y), z(z) {}
        ~Point3f();

        void set(float, float, float);
        void reset(void);
        void print(void) const;
        float distance(const Point3f&) const;

    }

The Main_handleKeyPressEvent() method is called in the Main_poll() method. The
Main_poll() method is called for each iteration of rendering.

void Main_poll()
{
    while (SDL_PollEvent(&mainEvent))
{ switch (mainEvent.type) {
    case SDL_KEYDOWN:
        // callback for handling keystrokes, arg is key pressed
        Main_handleKeyPressEvent(&mainEvent.key.keysym, mainEvent.key);
        break;
    }
}

Step 2 Get radius of curve

The tool used for determining the radius of the circle formed by the arc is Matlab. Before I can delve into the program, I would like to explain the algorithm for determining the radius. Consider the following Figure 4.1.
Figure 4.1. Elevated curve (SE) radius calculations.

SE – The curve that is rendered

YDIFF – The difference in y-coordinates of S and E

L – The linear distance between S and E
R – Radius of arc

C – Center of circle formed by arc SE

**Calculations for R**

\[
\sin(\alpha_1) = \frac{YDIFF}{L}
\]

\[
\therefore \ (\alpha_1) = \sin\left(\frac{YDIFF}{L}\right)
\]

The figure implies that

\[
\alpha_1 + \alpha_2 = 90^\circ = \frac{\pi}{2}
\]

\[
\therefore \ \alpha_2 = \frac{\pi}{2} - \alpha_1
\]

\[
\cos(\alpha_2) = \left(\frac{L}{2}\right) / R
\]

\[
\therefore \ R = \frac{L}{2\cos(\alpha_2)}
\]

The Matlab program that implements the above algorithm is,

```matlab
function R=radius(x,y)

ydiff=abs(x(2)-y(2));
ArcLength=sqrt((x(1)-y(1))^2+(x(2)-y(2))^2+(x(3)-y(3))^2);

angle1=asin(ydiff/ArcLength);
angle2=pi/2-angle1;

R=(ArcLength/2)/cos(angle2);
```

Set x and y coordinates to the start and end point of arc and run the above function R.
Example,

\[ x = [-157.42, -0.005, -2692.81]; \]
\[ y = [-142.24, 3.04, -2719.66]; \]
\[ R = \text{radius}(x, y) \]
\[ = 1.577384113300484e+002 \]

A precision of 4 decimal points is very accurate. Thus, the corresponding database line in `roads.txt` would look like,

1) Road ID – 5
2) Start point’s X coordinate – -157.42
3) Start point’s Y coordinate – -0.005
4) Start point’s Z coordinate – -2692.81
5) End point’s X coordinate – -142.24
6) End Point’s Y coordinate – 3.04
7) End Point’s Z coordinate – -2719.66
8) Width of the road – 21.214776
9) Radius of the circle formed by the curve – 157.7384
10) Texture image file – `cherry1new.rgb`
11) Number of segments – 15
12) The first segment to be rendered – 1
13) The last segment to be rendered – 15
14) The model of the road – 3
15) The difference between the start point’s y-coordinate and end-point’s y-coordinate.
A unique name for this segment of the road - govind@cherry444

Since the road is a pretty small segment, the number of segments of the curve can be anywhere between 8 and 15 for optimal performance. If it is greater than 15, the application takes a long time to load the road. Any number below 8 would make the road segment rugged and not smooth.

4.2 Other modifications to the environment

All the signs and trees had to be changed to new positions. This had to be done because; the previous road had different height and width. The new positions were again determined by driving through the environment and hitting the hotkey at the desired position. The same had to be done in the case of grass cover and curbs.

4.3 Limitation of implementing algorithm in the driving environment

The only limitation of this algorithm in the simulated environment generation is that the elevated roads can only be straight roads. There can be no curvature in the XZ-axis.
Chapter 5 Modifying the Autonomous Vehicle Database

The premise of an algorithm being as good as the input also applies to this. This algorithm is a bit more tedious to implement than the elevated curve algorithm since it needs the coordinates of the center point of the circle as input and not the radius. Thus, the key to the success of this algorithm and smooth path of autonomous vehicles is finding the correct center of the arc. The tool used to go about with modifying the curve is Autocad 2008.

5.1 Steps

There are 3 major steps in implementing this algorithm.

1) Get two straight line segments between which the arc has to be drawn.
2) Get a point approximately in the middle of the curved path. Plot this arc and note the radius.
3) Using the above radius, fillet the 2 straight lines.

Step 1 Get straight lines

Two arcs/curves adjacent to each other cannot be smooth unless the end of first arc points to the exact direction as the start of the next arc. This is possible only if the 2 arcs are part of a continuous arc. Refer to Figure 5.1 and 5.2 for the proof. Thus, an arc should be adjoining straight lines. To get the straight lines, one should drive through the environment and use the hotkey F2 to get the coordinates of interest.
Figure 5.1. Discrepancies with two contiguous curves.

Figure 5.2. Discrepancy solved by having a line segment.
Step 2 Plot approximate curve in Autocad

After plotting the straight lines, we have to plot a curve that approximately fits. To plot an arc we need 3 points on the curve. The start and end points (line segments) are already present. To get a point that is part of the curve, one need to drive through the environment and get the coordinates using hotkey F2.

One can plot an arc using various means which include, two (start and end) points and an angle/direction or center point. I choose the mid-point of the arc so that the vehicle follows the road because; roads/paths are not perfect curves.

Plot the arc in Autocad and note the radius of the arc.

Figure 5.3. Approximate arc.
The highlighted ovals in Figure 5.3 also show that the curve does not smoothly blend with the straight lines that it connects. The next step solves this.

**Step 3 Fillet**

This step deals with the fillet operation that outputs the exact curve with smooth transformation from straight line to curve.

In its simplest manifestation, fillet automatically adds an arc between two lines or curves, or any combination thereof, and trims or extends them as necessary. To try this, draw two non-parallel lines as shown in the left side of Figure 5.4.

![Figure 5.4. The Fillet command inserts a tangent arc and trims the lines accordingly.](image)

Now start the Fillet command by entering it at the Command prompt, or by selecting Modify\Fillet, or by clicking the Fillet button on the Modify toolbar. In any case, it brings up the following prompt:

Current settings: Mode = TRIM, Radius = 0.0000

Select first object or [Undo/Polyline/Radius/Trim/Multiple]:

51
The radius from the previous step should be given as input and the 2 straight lines have to be selected. Autocad creates the exact curve that makes the transition between adjacent path segments smooth.

Figure 5.5 demonstrates the arc of the previous step after the fillet operation has been completed.

![Figure 5.5. Curve after fillet operation.](image)

The blue arcs show the final curve. The highlighted ovals show that the arcs perfectly fit the lines. Seeing the properties of the arcs will give the respective center points as well as the
slightly modified start and end points. These can then be passed to the application. The start and end points are slightly modified to fit the best curve. Figure 5.6 shows that there is a slight change in the start and end coordinates too. The red arc is the approximated arc and blue arc is the filleted accurate arc.

For a curve with start point $(360.41, -116.91, 1.0)$ and end point $(382.8479, -105.2627, 1.0)$ the database line after the fillet operation would be

1) Vehicle Model Name – $\text{Model}Benz$ ($\$ \text{ indicates that the velocity is a multiple of main driver’s velocity}$

2) Vehicle ID – 21
3) Path type – CurvePath
4) Velocity – 0.5 (half the speed of main vehicle)
5) Acceleration – 0.0 (no acceleration)
6) Deceleration – 0.0 (no deceleration)
7) Start point’s X coordinate – 360.41
8) Start point’s Y coordinate – -116.91
9) Start point’s Z coordinate – 1.0
10) End point’s X coordinate – 382.8479
11) End point’s Y coordinate – -105.2627
12) End point’s Z coordinate – 1.0
13) Turn variable – clockwise
14) C’s X coordinate – 342.6855
15) C’s Y coordinate – -55.328
16) C’s Z coordinate – 1.0
17) Scale Factor – 1.0
18) Angle of rotation about the X-axis – -90.0
19) Angle of rotation about the Y-axis – 180.0
20) Angle of rotation about the Z-axis – 0.0
21) Indicator render variable – 0
22) Traffic-light Id – 0
Chapter 6 Summary and Future work

This chapter summarizes the algorithms and their implementation. There are two algorithms, one for elevated roads (hills) and the other for flat curves. There are two algorithms because the conditions for an elevated curve are different from that of a flat curve. An elevated curve takes into consideration that the road always starts or ends parallel or perpendicular to the y-axis.

6.1 Elevated curve

There are four types of curves possible with this algorithm – Flat-concave, Concave-flat, Flat-convex and Convex-flat. This algorithm consists of 3 steps:

1) Find Angle subtended by the arc at the center.

\[ \text{Angle} = \frac{\pi}{2} - \text{asin}\left(\frac{R - YDIFF}{R}\right) \]

WHERE,

R – Radius of circle formed by the arc.
YDIFF – The difference in y-value of start and end point of arc.

2) Find Angle Increment (\(\delta\)) and Angle subtended by curve on XZ-plane (\(\alpha\))

\[ \delta = \frac{\text{Angle}}{\text{numSegments}} \]

WHERE, numSegments = Number of Segments

\[ \alpha = \text{atan}\left(\frac{pe.z - ps.z}{pe.x - ps.x}\right) \]
3) For each angle increment,
   a. Calculate increments in x, y and z axes.
      
      y-axis increment = $R \times \cos(\delta) - R \times \cos(2\delta)$
      
      xz-plane increment = $R \times \sin(2\delta) - R \times \sin(\delta)$
      
      x-axis increment = $XZINCR \times \cos(\alpha)$
      
      z-axis increment = $XZINCR \times \sin(\alpha)$
      
   b. Pass new coordinates to another function that finds the four corners of the road.

Apart from implementing the above algorithm, the coordinates of trees, signs etc on the
created road segments should also be modified so that they are at the correct (ground) level.
Even the paths of the autonomous vehicles have to be changed in order to traverse the road. To
make the autonomous vehicle traversal smooth, its path values should be finely divided so that
the uneven changes in y-axis values are accounted for.

6.2 Flat curve

There are two models of curves possible with this algorithm; clockwise and counter-
clockwise. This algorithm involves the following steps in an iterative process till the end point
of the arc is attained/rendered:

1) Calculate angular increment ($\alpha$)

$$\alpha = \frac{L}{R}$$
WHERE,

\[ L = \text{velocity} \times \delta t + \frac{1}{2} \text{acceleration} \times (\delta t)^2 \]

R – Radius of circle formed by arc

2) Get initial displacement angle (\( \beta \)) between Center C and Start point S.

\[ \beta = \text{atan} \left( \frac{\text{Difference in z-coordinates of start and center points}}{\text{Difference in x-coordinates of start and center points}} \right) \]

3) Calculate displacement for new position(S1) after \( \alpha \) increment from the Center C along the X and Z axes

\[ S_{1.x} = C.x + R \times \sin(\beta + \alpha) \]

\[ S_{1.z} = C.z + R \times \cos(\beta + \alpha) \]

4) Calculate displacement for new position after \( \alpha \) increment from the Start point along the Y-axis

Change in y-axis = YDIFF * Displacement / arc length

WHERE

arc length = radius * (total angle subtended by arc at center)

Displacement = length of line segment SS1

YDIFF = difference in y-coordinates of start and end points of arc
5) Calculate rotation angle by converting the input angles into radians. This can be done by multiplying the value with \( \left( \frac{180}{\pi} \right) \).

6) Define new coordinates using the above calculations.

### 6.3 Future work and improvements

One improvement that can be imposed is on the elevated curves. The elevated curves are presently straight roads with hills. This model can be changed to include curves along the XZ-plane along with curve along the y-axis. This would make the elevated-curve algorithm to render flat-curves too. This would include a few complications since this algorithm would be a combination of both the aforementioned algorithms.
References


Achal, Oza (April 2006) Modeling Designer-Friendly Virtual Environments Using Splines, Master’s Thesis, Northeastern University, Boston MA


Joseph Kearney, Peter Willemsen (December 1999) Scenario Languages for Driving Simulation, University of IOWA, Iowa City, IOWA
Appendix A  Changes in the Virginia Project

The algorithms were implemented in the Virginia Project. This section defines the specific roads and approximate coordinates of the sections of the project that have been worked on.

The elevated curve algorithm was implemented on Cherry Avenue. This is the road connecting 6th Street and Ridge Avenue. The road data corresponding to this road can be found in Tile 12 of the data folder. The following are a few of the coordinates and database values corresponding to this road.

5 -507.869 18.288 -2293.2996 -431.57 12.01 -2366.28
21.214776890.9829 cherry1new.rgb govind@cherry111
5 -431.57 12.01 -2366.28 -250.89 -3.04 -2556.84 21.214776
2298.5009 cherry1new.rgb govind@cherry222
5 -172.89 -3.05 -2665.45 -157.42 -0.005 -2692.81
21.214776163.7376 cherry1new.rgb govind@cherry333
5 -157.42 -0.005 -2692.81 -142.24 3.04 -2719.66
21.214776157.7384 cherry1new.rgb govind@cherry444
5 -135.79 3.06 -2730.47 -117.48 4.95 -2761.33
21.214776341.5787 cherry1new.rgb govind@cherry555
5 -117.48 4.95 -2761.33 -95.440 6.84 -2798.45 21.214776
493.97569 cherry1new.rgb govind@cherry666

The flat-curve algorithm was implemented for the path of the lead car along Jefferson Park Avenue onto Old Lynchburg Road till 5th street. This was chosen because the lead car is
pretty close and slow thus making a jerky motion was very prominent. The autonomous vehicle data corresponding to the above mentioned lead car is present in Tile 1 of Scenario data folders of 1A and 1B. A few of the coordinates of the path of the vehicle are:

```
$pModelBenz, 21, StraightPath,0.6,0.0,0.0,295.7844,-135.5117,1.0,360.41,-
116.91,1.0, none,0.0,0.0,0.0,1.0, -90.0, 180.0,0.0,0.0,
$pModelBenz, 21, CurvePath,0.5,0.0,0.0,360.41,-116.91,1.0,382.8479,-105.2627,
1.0, clockwise,342.6855,-55.328,1.0,1.0, -90.0, 180.0,0.0,0.0, 
$pModelBenz, 21, StraightPath,0.5,0.0,0.0,382.8479,-105.2627,1.0,454.5849, -
48.0589, 1.0, none,0.0,0.0,0.0,1.0, -90.0, 180.0,0.0,0.0,
```

The data is just a part of the whole path of the lead car. The path of starts on Jefferson Park Avenue at (-54.05, -4.581, 1.0) and ends on 5th street at (1747.73, 411.28, 13.14).

To summarize, for autonomous vehicles to go smooth on flat-curves, there are a number of points to be noted. For a straight path, the start and end points are needed. For curved paths, start, end and a point in between are needed. These points have to be plotted and filleted to get proper curves. The time estimate for the whole process of getting a single curve’s correct coordinate is:

1) Get coordinates by driving through the environment and hitting F2 hotkey at appropriate points – 15 minutes

2) Plot all points (2 straight lines and 1 curve) in Autocad – 10 minutes

3) Fillet the straight lines after noting the radius of the above plotted curve – 5 minutes

4) Update the `auto_data.txt` to reflect the above change and drive through the environment to verify – 10 minutes

5) Repeat above steps the vehicle is still not smooth or is not going on the road.

Total time for 1 curve – 50 minutes
For coordinates on a hill, apart from the above mentioned steps, the resultant path has to be divided into number of segments in Autocad to take the uneven changes in height into consideration. This would take about 10 minutes for each segment.
Appendix B  Recommended Changes

The other significant roads those have to be changed using the elevated curve algorithm are Blenheim Avenue, Monticello Avenue near Interstate I64, Ramp from Monticello Avenue onto I64 and Ramp from 5th street onto I64.

The flat-curve algorithm can be used to implement all the paths of all the autonomous vehicles which include, vehicles on 5th Street, I64, Old Lynchburg Road, the initial lead car on Scenarios 2A and 2B, that goes on Wertland Avenue and continues on Main Street.