Dean O'Keeffe

BA Computer Visualisation and Animation
Year 3

Innovations Project

Autonomous Traffic Generation for Production Animation

Abstract

This report represents the research and development of an application that could be used for the creation of autonomous traffic in production animation.

The report begins by examining flocking systems and steering behaviours. An example of ambient traffic within a game environment, which uses some of these steering behaviours, is then examined. The report briefly discusses the advantages and disadvantages of implementing these behaviours into an autonomous traffic generator for production animation.

We then discusses the two possible languages that the application could be written in, Maya's MEL scripting language or the C programming language. Advantages and disadvantages are discussed for both approaches and a solution is reached.

The report then describes the methods used to represent a road network and how vehicles are controlled around that network. The report then explains how these methods are implemented within a C program. The fully commented source code is also included.

Finally, the report examines the success of the application and explains how the application could be developed further.

Contents Page

The Concept

The animation of a street scene in a production may require the presence of ambient traffic. This ambient traffic could be manually animated by an animator and this would be a relatively simple task. Consider that this scene then increases in scale, or that the density of traffic increases, so that the street scene scales to a full city environment. The animator's job has become highly complex. Not only must the animator consider the movement of the car, but the interaction of all the cars within the scene. Each car influences every other car and it can be said that there is an exponential increase in the complexity of the animation. There is now a substantial amount of planning and keyframing required by the animator.

The problem becomes even more troublesome when changes are required in the traffic. For example, consider that the traffic has been manually animated and each car has been keyframed, a decision is then made that one of the cars needs to be removed from the scene. This would almost certainly result in the animation falling apart. A car that had been animated to approach a junction and queue behind the removed car would now look peculiar, as there would be an open space in front of it. Traffic may also wait at junctions when the road was clear because previously the removed vehicle had been travelling along that road. More problematic would be the insertion of extra cars, as all other cars would need re-keyframing to avoid collisions that cause them to pass through one another. To summarise, the whole scene would require re-planning and re-animating every time the volume of traffic was adjusted. This is an inefficient use of an animator's time and would be a tedious task.

This project examines the various techniques available that could be used to create autonomous traffic for production animation. A selection of these techniques are then interpreted and implemented to create a product that would allow the generation of complex road networks. The traffic in these road networks would be aware of other vehicles and react accordingly. For example, traffic would queue at junctions. The traffic in the road network would also be controlled by a traffic light system at junctions. The most important aspect of the final product is adaptability, the product must be scaleable so that more complex roads such as roundabouts, T-junctions and slip roads can be constructed and implemented without drastic reconstruction of the product.

Research

The first area of research was flocking systems. Papers produced by Craig Reynolds became the major reference point for this area of research.

> The basic flocking model consists of three simple steering behaviours which describe how an individual boid manoeuvres based on the positions and velocities its nearby flockmates; separation, alignment and cohesion. Separation steers to avoid crowding local flockmates, alignment steers towards the average heading of local flockmates and cohesions steer to move toward the average position of local flockmates [1]

As a driver and after observing traffic I would certainly support the theory that vehicles also adhere to these rules at some level. Cars will maintain a certain amount of separation when travelling. In the case of the flow of heavy traffic in one direction this would be, on average, a two second difference in passing a fixed point. This creates a variation in the separation distance that is dependent on the speed of the vehicles. In reality this is to take into account braking distances and reaction times. Traffic flowing in opposite directions will, in the UK at least, maintain separation by keeping to the left, thereby avoiding head on collisions. This is linked into alignment, cars will align themselves in traffic but this is due mainly to the fact that the cars follow roads and road markings and cannot venture off them in normal circumstances. Cohesion in cars is actually not desired in traffic as this usually means that there is some form of traffic jam, cars will tend to want to take separate routes to avoid bottlenecks. Cars do not act as a flock over time; each car will have some form of target location that will be independent of other traffic, which will result in the car leaving the flock. Cars will tend to only act as a flock on a micro scale, when travelling along a road amongst heavy traffic for example.

To conclude this area of research it can be said that flocks would allow the representation of traffic at some level but there would need to be some adaptation to create a realistic traffic system. This led to research into more advanced steering behaviours for autonomous characters. Once again Craig Reynolds had written the most definitive paper on this area of flocking systems.

> Steering behaviours for autonomous characters presents a collection of simple, common steering behaviours which includes seek, flee, pursuit, evasion, offset pursuit, arrival, obstacle avoidance, wander, path following, wall following, containment, flow field following, unaligned collision avoidance, separation, cohesion, alignment, flocking, and leader following.[2]

Of particular interest and relevance to the creation of autonomous traffic would be obstacle avoidance, path following, wall following, containment, separation, cohesion and alignment. The latter three have already been discussed in relation to autonomous traffic. Here follows my interpretation of how the remaining steering behaviours could be implemented into autonomous traffic generation. Obstacle avoidance has an obvious contribution to offer for traffic simulation. Traffic cannot be allowed to collide, unless this is the desired effect, so some form of obstacle avoidance must be implemented. In reality, traffic will avoid obstacles such as walls but we can also consider other traffic to be obstacles, and this will determine how traffic flows. For example, at a busy slip road a bottleneck may occur as traffic flow slows to avoid collisions. Path following could be implemented as all possible routes for the traffic could be pre-calculated. Traffic then chooses which path to follow at predetermined points, such as junctions. Wall following may be useful as the traffic is required to stay on the left hand side of the road. Wall following would be one way to implement this.

The problem with this approach is the summing of these simple behaviour parts to create the more complex traffic whole. This is discussed, and several different approaches are suggested, in Reynolds paper[2]. This approach would have been possible, but would have lead to a very large area of research and experimentation. It was felt that other more simplified solutions could be found and this lead to research in into how games developers approach these problems.

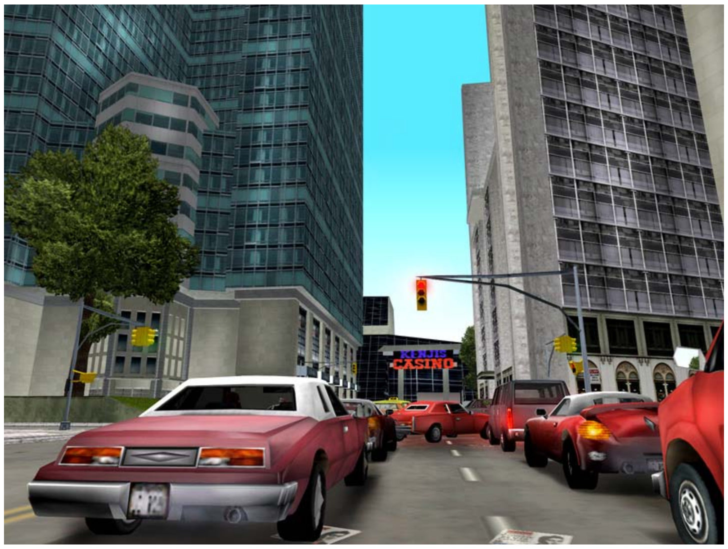

Figure 1. Traffic queuing at a traffic light junction in GTA3.

Modern games are certainly very sophisticated, but limitations still exist for games developers in terms of the amount of memory and processing power available. This results in developers approaching problems in creative ways, ensuring that they get the most out of the platform they are developing for. Games such as Grand Theft Auto 3 (Figure 1) and Midtown Madness (Figure 2) are amongst a collection of recent games that feature real-time ambient traffic within the in game environment. Fortunately Angel Studios, the developer for Midtown Madness, have produced an excellent article discussing the techniques employed for the creation of ambient traffic[3]. As I understand the paper, Angel Studios have employed a path following technique for the cars, some of these paths are predetermined but some are plotted on the fly.

> During normal driving conditions, all the ambient vehicles are positioned and oriented by a 2D spline curve. This curve defines the exact route the ambient traffic will drive in the XZ-plane.[3]

Since the lanes for ambient vehicles on each road are defined by a list of vertices, a road sub segment can easily be created between each vertex in the list. When the ambient vehicle moves from one segment to the next, a new spline is calculated to define the path the vehicle will take. Splines are also used for creating recovery routes back to the main rail data. These recovery routes are necessary for recovering the path after a collision or a player-avoidance action sent the ambient vehicle off the rail. Using splines enables the ambient vehicles to drive smoothly through curves typically made up of many small road segments and intersections.[3]

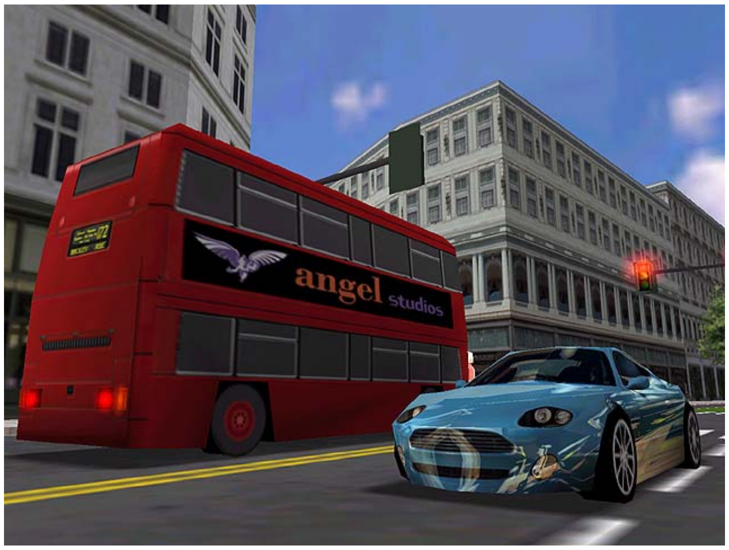

Figure 2. A screenshot from Midtown Madness showing two of a variety of vehicles that can be found amongst the ambient traffic.

This approach certainly works, but it has to meet requirements for the game that are not required for production animation. Although the cars follow paths, they can be taken off these paths during collisions as the player interacts with the game. The ambient traffic must always be ready to plot a new path back to the predetermined paths at any time. This is not required for the production animation system as all traffic activity can be predicted.

One last area of research worth mentioning is engineering projects. Traffic simulation is a valuable tool when it comes to town and city road planning. The opportunity to test new road networks in a virtual environment allows groups to see systems working and refine them before spending large amounts of money on building them. Unfortunately there are few papers on this area that contain relevant information or that reveal useful ideas. The majority of papers are old and most websites are dead links. This may be because these projects have a high monetary value, and it is understandable that companies and individual are protective of their latest techniques.

In summary we looked at flocking systems and steering behaviours for flocking systems. Research was then concentrated on the games development and it was discovered that one developer had employed a path following technique in the creation of interactive ambient traffic.

Development Environment

Two development environments were considered for the implementation of autonomous traffic generation. These were MEL, the Maya Embedded Language, and the C programming language. Both approaches had advantages and disadvantages, which we shall cover, that influenced the final decision.

The major advantage of MEL is that it allows full access to Maya's capabilities. For example, particles along with any other object within Maya can be created, animated and keyframed using a MEL script. Anything that can be achieved using Maya's graphical user interface can be accessed through MEL. This is a huge advantage in development as ideas can be tested manually using the graphical user interface before being incorporated within the MEL script. An additional advantage of MEL is that it is a scripting language. This avoids compiling and linking of the code before runtime, consequently results can be seen immediately and are evaluated in real-time.

Unfortunately MEL is not the most ideal development environment as the development tools available are both limited and weak. A good example of this would be the lack of debugging tools. This makes it harder to track and construct a large well structured script, this is a major disadvantage.

Another disadvantage of using MEL is that it does not support pointers, although values are passed by reference to functions. This could restrict a programs design. Additionally MEL does not appear to support structures, which are used to store a collection of variables that can differ in type. MEL can only offer arrays, which can only hold variables of the same type. This could again restrict the programs structure.

The C programming language by comparison is widely supported, well documented and has a large quantity of tools available to aid the development of applications. Structures and pointers are available as well as many other data structures. By using Microsoft Visual C++ 6 errors, such as syntax errors, can be easily detected in the code and crashes can be avoided by running compiled applications through a debugging tool. Visual C++ also allows the program to be displayed in a clear and structured manner and allows the user to access members of structures using shortcuts, speeding up code input and avoiding typographical errors.

The disadvantage of using the C programming language is that there is no high level graphics library that could rival Maya's rendered output available. Since the autonomous traffic is for production animation this presents a problem. My solution to this was to have the C program output MEL commands to a MEL file. This MEL file can then be loaded into Maya to control vehicle objects within the scene. This pipeline is shown in Figure 3.

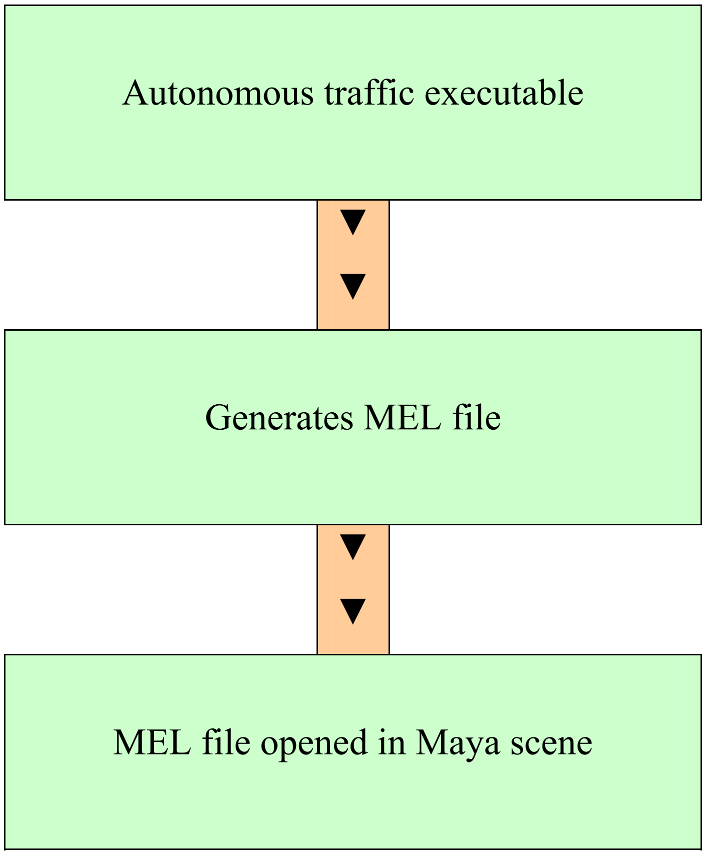

Figure 3. Pipeline of autonomous traffic generation.

This solution offered the best of both environments, the well supported development environment of the C programming language and the advanced capabilities of Maya that could be accessed by MEL.

The Program

The autonomous traffic generation application is based around the idea of location nodes. This basically means the road network is populated by a series of location nodes. Location nodes represent points where cars can travel to on the road network. The process begins by building the road network scene within Maya. Cubes, which are used to represent the location nodes, are positioned in the scene in such a way that a vehicle could travel between them throughout the road network without leaving the road or colliding with obstacles. These cubes are invisible within the scene, and are only used to plan the layout of the location nodes and gain X, Y and Z locations within the scene. Figure 4 shows how a series of location nodes can be used to represent the routes through a bend in a road. Also note how location nodes representing the path of traffic on opposing sides of the road cannot be allowed to cross as this would result in a collision.

The location nodes are also allocated numbers which allows a list to be formed that represents connected nodes. For example, Figure 5 shows how each location node can have a maximum of three location nodes as the next location node. From location node 1 a vehicle has a choice to travel to node 2, node 3 or node 4. If there is no next node the next node member of the location node is given a null value.

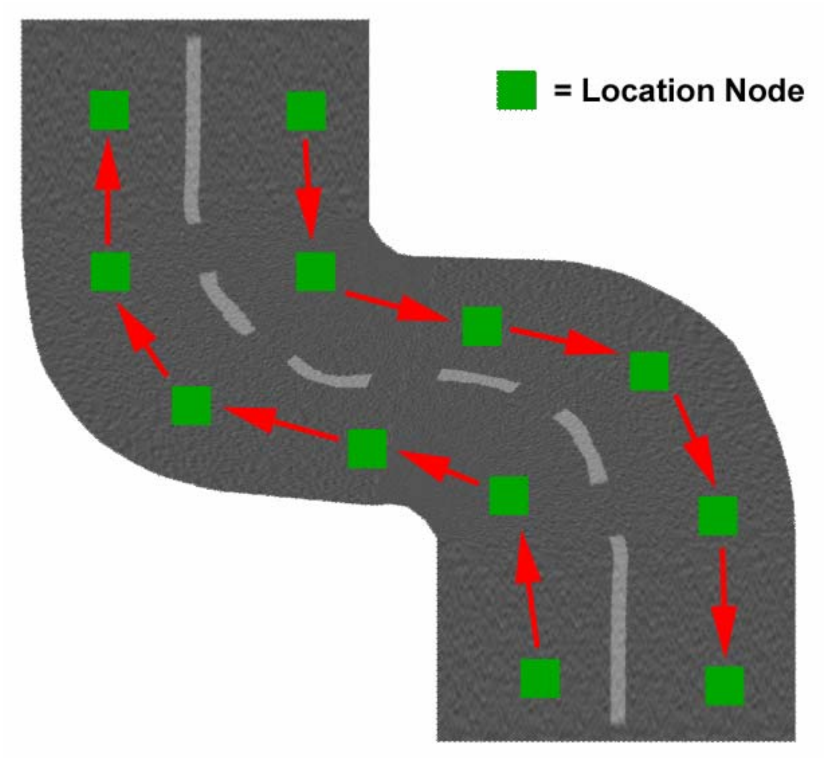

Figure 4. Nodes can be positioned to allow cars to travel around bends.

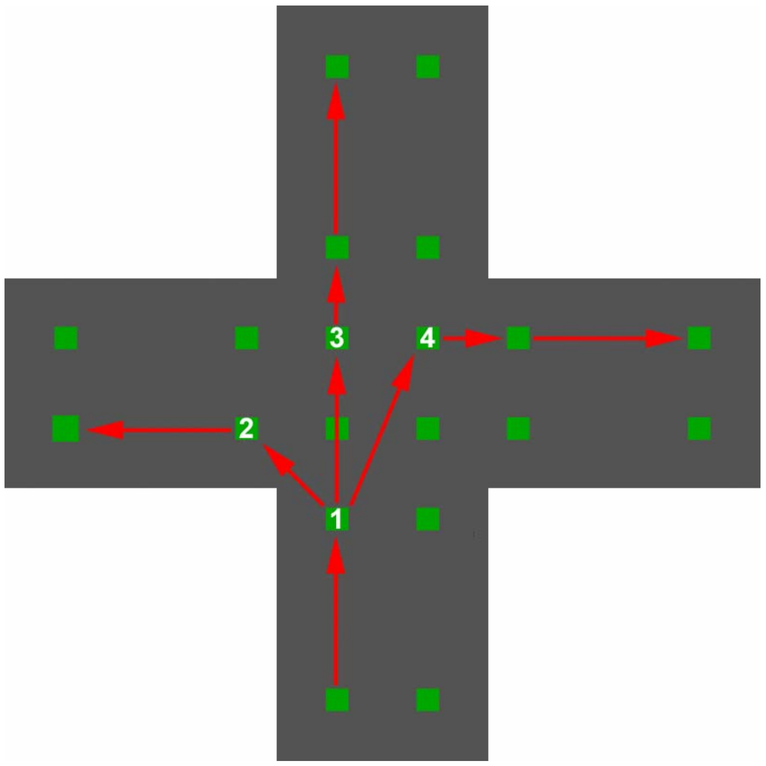

Figure 5. Connected nodes through a four way junction.

Location nodes also have two other tasks. Location nodes have a status attribute that indicates whether a vehicle occupies that location. This is to avoid having two or more cars at one location, which would constitute a collision. The second attribute indicates whether a vehicle located at that node has permission to move to the next node. This is used to represent traffic lights, if there is a red light then the permission of that location node will stop a vehicle from moving off. Permission could be used for any circumstance where vehicles are required to stop, such as normal junctions or pelican crossings, but these would require independent algorithms to set the permissions.

The location node data structure must hold the following information:

> Next node 1, Next node 2, Next node 3
> Status
> Permission
> X, Y, Z

The next structure is the car node. The car node represents the car travelling around the network of nodes. The car node holds a value that indicates which location node the car is currently located at. The car node also holds an X, Y and Z value that is fetched from the current location node.

The car node data structure holds the following information:

> Current location node
> X, Y, Z

The program then has the responsibility to maintain the car and location nodes whilst moving the car nodes around the location nodes. To demonstrate this and prove that this method works I created an application was created that controlled 4 car nodes across a four way traffic light junction. I chose this type of junction because it can be connected to other four way junctions easily to create a larger road network. The basic program flow can be seen in figure 6.

The program begins by setting the car nodes in start positions. This is done by giving the car node member a location node value. The next stage is to initialise the permissions of the location nodes that represent the traffic lights, and this is a function that we shall call change lights. We must ensure that only one location node has its permission set to 0, so that only cars waiting at that node can proceed across the junction. The change lights function randomly sets one of the junction nodes permission to 0, and changes the other location node permissions to 1.

The next stage is a for loop through the frames, starting at frame 0 and incrementing by 50 frames at a time. The program prints the new frame number to a MEL file in the following way:

```
    fprintf(cfPtr, “currentTime %dpal;\n”, frame);
```

Which, if the value of frame was currently 150, would read in the MEL file as:

```
    currentTime 150pal;
```

When opened in Maya’s script editor this would set the current frame to 150.

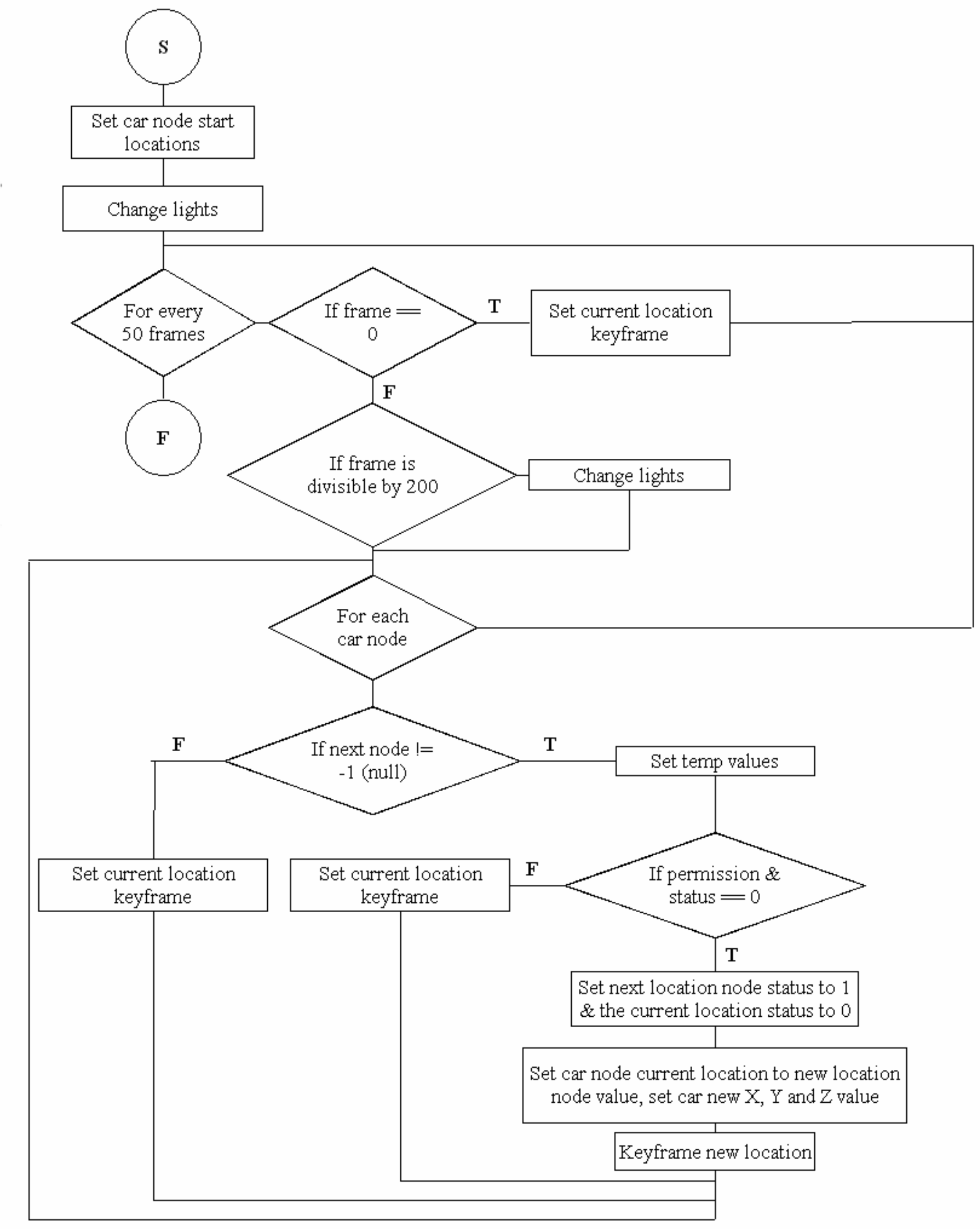
S
Set car node start locations
Change lights
For every 50 frames
F
If frame == 0
T
Set current location keyframe
F
If frame is divisible by 200
Change lights
For each car node
F
If next node != -1 (null)
T
Set temp values
Set current location keyframe
Set current location keyframe
F
If permission & status == 0
T
Set next location node status to 1 & the current location status to 0
Set car node current location to new location node value, set car new X, Y and Z value
Keyframe new location

Figure 6. Program flow of the autonomous traffic generation application

If the current frame is equal to 0 the application will set the keyframes for the car nodes current locations, which will be their start locations. Keyframes are also set by printing to a MEL file.

```
        fprintf(cfPtr, “setKeyframe –value %d car_%d.translateX;\n”,
                car[carCount].x, carCount);
```

This is repeated twice more, once for the Y value and once for the Z. If the current car node was car node 3 and it’s X component was 20, the above statement would read in the MEL file as.

```
    setKeyframe –value 20 car_3.translateX;
```

When opened in Maya’s script editor this would translate an object named car_3 by 20 in the X axis and then set a keyframe. This is where it is important to have set the current time, as the keyframe is set at the current time. Failure to set the current time would give strange results.

The next stage is to see if the current frame is divisible by 200. If it is, then the change lights function is called. This means that the traffic lights change every 200 frames. The traffic lights change in sequence, which allows each junction location node to have its permission set to 0 once every 600 frames.

The application then loops through each car. The first step is to determine whether the cars next node is valid. If the cars next node has a null value, represented by -1, then the car cannot move to that node. The car is then keyframed in its current location in the same way as previously described.

If the car has at least one valid next node then this is assigned to a temporary variable. The next node is decided by calling the next node function. This function determines whether the location node has more than one next location. If it has, the function randomly selects one and returns it to a temporary variable.

However, if the next node location is already occupied by a car, the status member of the location node has a value of 1, or the current location node has its permission set to 1, the car cannot be allowed to move. The car is then keyframed in its current location.

If the permission and the status indicate that the car can be allowed to move then the following changes are made. The current location node's status is set to 0. This is because the car will no longer be occupying this location. The next location node's status is then set to 1, since the car will now be occupying this location. The car node's current location is assigned the value of the next location node, which now becomes the current node. Finally the car's X, Y and Z values are assigned the X, Y and Z values of the new location node. The car's new X, Y and Z values are then keyframed in the usual manner.

This process is looped for as many frames as the user wishes, but since this application controls cars within a small junction it only loops through 1000 frames. This is more than adequate for the cars to reach their final positions.

For the code implementations of this program please refer to appendix 1, which contains the fully commented C code.

Conclusion

The program successfully controlled traffic travelling through a four way traffic light junction. The application successfully generates a MEL file that can be loaded into a Maya scene. This proves that this technique can be used to create autonomous traffic. There is no reason why the scale of the application could not be increased incorporate and control a larger road network.

Using the location node method we achieved a high level of control over the movement of the cars. This avoided unexpected results that could have resulted from using a behavioural flocking system, which relies on the agents calculating their behaviours correctly. The location node system also offers a simple built in collision avoidance system. This avoided the need for additional collision detection that could have been problematic in implementation as well as time consuming because of the additional research required.

Adaptability was one of the requirements of the autonomous vehicle generation application, this has been achieved. The location node method allows for the inclusion of a hilly road network since the location nodes contain the X, Y and Z components of the location in the scene. This method allows the car's X, Y and Z location in the scene to be keyframed, so the car can move up and down. Secondly, any junction or road layout could be represented by using location nodes; all that is required is a function for each junction type which would control the location nodes within that junction.

The first disadvantage of using location nodes is that the user must manually determine how the location nodes connect to each other and then set the location nodes within the C code. This was fairly simple for a single four way junction but the number of location nodes increases rapidly when trying to construct a larger road network. There are two possible solutions to this problem. The first solution would be to construct each road and junction independently. Groups of location nodes that represent junctions and roads could then be connected like building blocks. The only task is to then renumber the location nodes, which should be a simple operation. The second solution would be more complex, and would require an algorithm to determine a path between nodes automatically. Path finding algorithms are available, but this would require a large amount of additional research.

An obvious disadvantage of the current application is that the motion of the vehicles is very simple. A solution to this could be to increase the number of location nodes. This solution makes the previous disadvantage even more apparent. More location nodes require more work to determine paths. Looking back to the research stage a solution may be to implement the same method Angel Studios employed for smooth motion in Midtown Madness. This solution would involve a spline curve being drawn between two nodes, with the vehicle then following this curve. It has been proved by Angel Studios that this method gives some very realistic results.

By implementing three of these solutions we could create a very advanced car motion. These solutions are to increase the number of location nodes, implement a pathfinding algorithm to determine a route through connected nodes and finally to draw a curve through these location nodes. This would allow vehicles to move around obstacles in the road, such as a parked car, since there would be more than one location node that the vehicle could pass through.

These solutions would be extremely difficult, although not impossible, to incorporate into the current system of the program writing to a MEL file. The solution to this could be to rewrite the application in C++. C++ offers even more facilities for better structured applications than C, which in turn offers more flexibility and would allow the application to grow. The second step would be to integrate this application into the Maya C++ API. The C++ API offers even more control within Maya than MEL. This allows the application to work within Maya. This should make it easier to implement the drawing of curves for example, since Maya can already do this. This would therefore help the user since they would not need to program this functionality.

Overall it can be said that the project was successful. The application proves that the idea of using location nodes works. However, the animations created are not to the standard that would be required for a production animation. This would require the integration of the more advanced solutions discussed.

Appendix 1

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


typedef struct {                                   /*  Location node structure definition  */
                int nextNode_01, nextNode_02, nextNode_03;
                int status, permission, x, y, z;
                } locationNode;



typedef struct {                                             /*  Car structure definition  */
                int currentNode;
                int x, y, z;
        } carNode;



void CarStartLocations(locationNode *, carNode *);           /*  Function declarations  */
void ChangeLights(locationNode *);
int NextNodeFunc(locationNode *, int);

int main(int argc, char* argv[])
{

        FILE *cfPtr = fopen("InnovationsOutput.mel", "a");            /*  File output setup  */





        int group = 1;
        int frame = 0, translate = 0;                        /*  Variable declarations  */
        int vehicle = 1;
        int lightChange = 0;
        int carCount = 0;
        int temp_currentNode = 0;
        int temp_nextNode = 0;


        srand(time(NULL));                         /*  Seed the random number generator  */

        locationNode node[20];                                        /*  Node definitions  */
        carNode car[4];
```

```
node[0].nextNode_01 = -1;            /*  location node initialisation  */
node[0].nextNode_02 = -1;
node[0].nextNode_03 = -1;
node[0].status = 0;
node[0].permission = 0;
node[0].x = 0;
node[0].y = 0;
node[0].z = -6;



node[1].nextNode_01 = 3;
node[1].nextNode_02 = -1;
node[1].nextNode_03 = -1;
node[1].status = 1;
node[1].permission = 0;
node[1].x = 2;
node[1].y = 0;
node[1].z = -6;



node[2].nextNode_01 = 0;
node[2].nextNode_02 = -1;
node[2].nextNode_03 = -1;
node[2].status = 0;
node[2].permission = 0;
node[2].x = 0;
node[2].y = 0;
node[2].z = -2;



node[3].nextNode_01 = 8;
node[3].nextNode_02 = 13;
node[3].nextNode_03 = 12;
node[3].status = 0;
node[3].permission = 1;
node[3].x = 2;
node[3].y = 0;
node[3].z = -2;



node[4].nextNode_01 = 5;
node[4].nextNode_02 = -1;
node[4].nextNode_03 = -1;
node[4].status = 1;
node[4].permission = 0;
node[4].x = -6;
node[4].y = 0;
node[4].z = 0;



node[5].nextNode_01 = 2;
```

```
node[5].nextNode_02 = 7;
node[5].nextNode_03 = 13;
node[5].status = 0;
node[5].permission = 1;
node[5].x = -2;
node[5].y = 0;
node[5].z = 0;



node[6].nextNode_01 = 2;
node[6].nextNode_02 = -1;
node[6].nextNode_03 = -1;
node[6].status = 0;
node[6].permission = 0;
node[6].x = 0;
node[6].y = 0;
node[6].z = 0;



node[7].nextNode_01 = 8;
node[7].nextNode_02 = -1;
node[7].nextNode_03 = -1;
node[7].status = 0;
node[7].permission = 0;
node[7].x = 2;
node[7].y = 0;
node[7].z = 0;



node[8].nextNode_01 = 9;
node[8].nextNode_02 = -1;
node[8].nextNode_03 = -1;
node[8].status = 0;
node[8].permission = 0;
node[8].x = 4;
node[8].y = 0;
node[8].z = 0;



node[9].nextNode_01 = -1;
node[9].nextNode_02 = -1;
node[9].nextNode_03 = -1;
node[9].status = 0;
node[9].permission = 0;
node[9].x = 8;
node[9].y = 0;
node[9].z = 0;



node[10].nextNode_01 = -1;
node[10].nextNode_02 = -1;
```

```
node[10].nextNode_03 = -1;
node[10].status = 0;
node[10].permission = 0;
node[10].x = -6;
node[10].y = 0;
node[10].z = 2;



node[11].nextNode_01 = 10;
node[11].nextNode_02 = -1;
node[11].nextNode_03 = -1;
node[11].status = 0;
node[11].permission = 0;
node[11].x = -2;
node[11].y = 0;
node[11].z = 2;



node[12].nextNode_01 = 11;
node[12].nextNode_02 = -1;
node[12].nextNode_03 = -1;
node[12].status = 0;
node[12].permission = 0;
node[12].x = 0;
node[12].y = 0;
node[12].z = 2;



node[13].nextNode_01 = 17;
node[13].nextNode_02 = -1;
node[13].nextNode_03 = -1;
node[13].status = 0;
node[13].permission = 0;
node[13].x = 2;
node[13].y = 0;
node[13].z = 2;



node[14].nextNode_01 = 17;
node[14].nextNode_02 = 12;
node[14].nextNode_03 = 6;
node[14].status = 0;
node[14].permission = 1;
node[14].x = 4;
node[14].y = 0;
node[14].z = 2;



node[15].nextNode_01 = 14;
node[15].nextNode_02 = -1;
node[15].nextNode_03 = -1;
```

```
                    node[15].status = 1;
                    node[15].permission = 0;
                    node[15].x = 8;
                    node[15].y = 0;
                    node[15].z = 2;



                    node[16].nextNode_01 = 11;
                    node[16].nextNode_02 = 6;
                    node[16].nextNode_03 = 7;
                    node[16].status = 0;
                    node[16].permission = 0;
                    node[16].x = 0;
                    node[16].y = 0;
                    node[16].z = 4;



                    node[17].nextNode_01 = 19;
                    node[17].nextNode_02 = -1;
                    node[17].nextNode_03 = -1;
                    node[17].status = 0;
                    node[17].permission = 0;
                    node[17].x = 2;
                    node[17].y = 0;
                    node[17].z = 4;



                    node[18].nextNode_01 = 16;
                    node[18].nextNode_02 = -1;
                    node[18].nextNode_03 = -1;
                    node[18].status = 1;
                    node[18].permission = 0;
                    node[18].x = 0;
                    node[18].y = 0;
                    node[18].z = 8;



                    node[19].nextNode_01 = -1;
                    node[19].nextNode_02 = -1;
                    node[19].nextNode_03 = -1;
                    node[19].status = 0;
                    node[19].permission = 0;
                    node[19].x = 2;
                    node[19].y = 0;
                    node[19].z = 8;

        CarStartLocations(node, car);          /*  Assign car start locations  */

        ChangeLights(node);                    /*  Assign traffic light node permissions  */
```

```
                    for(frame=0; frame<=1000; frame+=50){

                    /*  Increment through frames, 50 at a time  */


                              fprintf(cfPtr, "currentTime %dpal;\n", frame);

                              if(frame == 0){                                   /*  If the current frame is 0  */


          for(carCount= 0; carCount<=3; carCount++){           /*  Increment through each car  */


                                                               /*  Keyframe car's start position  */

fprintf(cfPtr, "setKeyframe -value %d car_%d.translateX;\n", car[carCount].x, carCount);
fprintf(cfPtr, "setKeyframe -value %d car_%d.translateY;\n", car[carCount].y, carCount);
fprintf(cfPtr, "setKeyframe -value %d car_%d.translateZ;\n", car[carCount].z, carCount);

                                        }
                              }

                              else

                                        if(frame >=50){


                                        lightChange = frame % 200;

                    /*  Check if 200 frames have passed, which would result in lightChange equalling 0  */

                                        if(lightChange==0)

                    /*  If 200 frames have passed, change traffic light node permission values  */

                                                  ChangeLights(node);

                                        for(carCount= 0; carCount<=3; carCount++){

                    /* Increment through each car  */


          temp_currentNode = car[carCount].currentNode;        /*  Assign temporary current node value  */

                                                               /*  Assign temporary next node value  */

          temp_nextNode = NextNodeFunc(node, temp_currentNode);


                                                  if(temp_nextNode != -1){

                    /*  If the nextNode is equal to -1, then it is null  */
```

```
            if(node[temp_currentNode].permission == 0 && node[temp_nextNode].status == 0){
            /*  if the current location node's permission and status are equal to 0  */



node[temp_currentNode].status = 0;

            /*  Change the current location node's status to 0  */

car[carCount].currentNode = temp_nextNode;

            /*  Change car's current location node to the current location node's next node  */
                                                            temp_currentNode =
car[carCount].currentNode;

            /*  Assign temporary current node with new current node value  */

car[carCount].x = node[temp_currentNode].x;

            /*  Assign new x, y, z values to the car node  */

car[carCount].y = node[temp_currentNode].y;

car[carCount].z = node[temp_currentNode].z;
node[temp_currentNode].status = 1;

            /*  Change the current location node's status to 1  */


fprintf(cfPtr, "setKeyframe -value %d car_%d.translateX;\n", car[carCount].x, carCount);
fprintf(cfPtr, "setKeyframe -value %d car_%d.translateY;\n", car[carCount].y, carCount);
fprintf(cfPtr, "setKeyframe -value %d car_%d.translateZ;\n", car[carCount].z, carCount);


                                                            }

                                                    else {

fprintf(cfPtr, "setKeyframe -value %d car_%d.translateX;\n", car[carCount].x, carCount);

/*  If the car cannot be moved, keyframe the car's current location  */

fprintf(cfPtr, "setKeyframe -value %d car_%d.translateY;\n", car[carCount].y, carCount);
fprintf(cfPtr, "setKeyframe -value %d car_%d.translateZ;\n", car[carCount].z, carCount);

                                                    }

                                            }

                                    }
                                    }

                    }
```

```
printf("Keyframe generation complete\n\nLoad the InnovationsOutput.mel file into the Maya script
editor\n\n");

                return 0;
        }




                void CarStartLocations(locationNode *node, carNode *car)

        /*  Pre set car start locations  */
          {



                        car[0].currentNode = 1;
                        car[0].x = node[1].x;
                        car[0].y = node[1].y;
                        car[0].z = node[1].z;


                        car[1].currentNode = 4;
                        car[1].x = node[4].x;
                        car[1].y = node[4].y;
                        car[1].z = node[4].z;

                        car[2].currentNode = 15;
                        car[2].x = node[15].x;
                        car[2].y = node[15].y;
                        car[2].z = node[15].z;

                        car[3].currentNode = 18;
                        car[3].x = node[18].x;
                        car[3].y = node[18].y;
                        car[3].z = node[18].z;




                }




                void ChangeLights(locationNode *node)

        /*  Change traffic lights  */
          {
                        if(node[3].permission == 0){
                                node[3].permission = 1;
                                node[5].permission = 1;
                                node[14].permission = 1;
                                node[16].permission = 0;
                                }
```

```
                    else
                            if(node[16].permission == 0){
                                    node[3].permission = 1;
                                    node[5].permission = 1;
                                    node[14].permission = 0;
                                    node[16].permission = 1;
                                    }

                                    else
                                            if(node[14].permission == 0){
                                                    node[3].permission = 1;
                                                    node[5].permission = 0;
                                                    node[14].permission = 1;
                                                    node[16].permission = 1;
                                                    }

                                                    else{
                                                            node[3].permission = 0;
                                                            node[5].permission = 1;
                                                            node[14].permission = 1;
                                                            node[16].permission = 1;
                                                            }
}


        int NextNodeFunc(locationNode *node, int temp_currentNode)

    /*  Random next location node choice  */
        {
            int choice;

            if(node[temp_currentNode].nextNode_02 == -1)

                    return node[temp_currentNode].nextNode_01;

            else
            {
                    choice = 1 + (rand() % 3);

                    switch (choice){

                    case 1:

                            return node[temp_currentNode].nextNode_03;

                    case 2:

                            return node[temp_currentNode].nextNode_02;

                    case 3:

                            return node[temp_currentNode].nextNode_01;

                    }
            }
}
```

CD Contents

PDF version of this report

Word version of this report

Readme instructions

Maya scene

Executable version of application

Source code

CD Instructions

1 Copy the Traffic_App folder to the computers hard disk.

2 Execute the application (Traffic_App.exe), this will create a MEL script.

3 Open the Maya scene (FourWayJunction.mb).
NOTE: The cars are represented by polygon planes with car symbol texture. This texture may not load into the scene. The texture can be reloaded through the Hypershade. The texture must be loaded into lambert 5. The texture can be located in the Symbols folder on the CD.

4 Load the InnovationsOutput.mel file into the script editor.

5 Press return.

6 Press play, the car symbols should move through the junction.

References

1 Boids (Flocks, herds and schools: A distributed behavioural model), Craig Reynolds, 1987, http://red3d.com/cwr/boids/

2 Steering Behaviours for Autonomous Characters, Craig Reynolds, 1999, http://red3d.com/cwr/steer/gdc99/

3 AI Madness: Using AI to Bring Open-City Racing to Life, Joe Adzima, Gamasutra 2001, http://www.gamasutra.com/features/20010124/adzima_pfv.htm

Figure 1. Midtown Madness 2, Angel Studios, Microsoft, http://www.microsoft.com/games/midtown2/multimedia.asp

Figure 2. Grand Theft Auto 3, Rockstar Games, http://www.rockstargames.com/grandtheftauto3/flash/main.html

Bibliography

AI Madness: Using AI to Bring Open-City Racing to Life, Joe Adzima, Gamasutra 2001, http://www.gamasutra.com/features/20010124/adzima_pfv.htm

Boids (Flocks, herds and schools: A distributed behavioural model), Craig Reynolds, 1987, http://red3d.com/cwr/boids/

GamaSutra.com, http://www.gamasutra.com/

Generation of ambient traffic for real-time driving simulation, Esmail Bonarkdarian, James Cremer, Joseph Kearney, Pete Willemsen, 1998, http://citeseer.nj.nec.com/cache/papers/cs/602/http:zSzzSzwww.cs.uiowa.eduzSz~cremerzSzpaperszSzimage98.pdf/bonakdarian98generation.pdf

Steering Behaviours for Autonomous Characters, Craig Reynolds, 1999, http://red3d.com/cwr/steer/gdc99/

Acknowledgements

Thanks to Stephen Bell and Phill Allen.