

An Experiment on Using Light to Affect Shortest-Path Calculations

by James Cookson

FILE	17805_JAMES_COOKSON_AN_EXPERIMENT_ON_USING_LIGHT_TO_AFFECT_SHORTEST-PATH_CALCULATIONS_78196_985774748.DOCX (413.67K)		
TIME SUBMITTED	21-APR-2016 01:41PM	WORD COUNT	8286
SUBMISSION ID	56200446	CHARACTER COUNT	43535



PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David

SWANSEA - ABERTAWE

An Experiment on Using Light to Affect Shortest-Path Calculations

By

James Cookson

Supervisor: Mr Gordon Dickers

Second Marker: Mr Andy Baker

Project submitted as part of the requirements for the
award of BSc(Hons) Computer Games Development

21st April 2016

DECLARATION

I, James Cookson declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed :

Date :

FORM OF CONSENT

I James Cookson hereby consent that my Project, submitted in candidature for the B.Sc.(Hons) Computer Games Development degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed :

Date :

Abstract

Lighting and pathfinding are two very common algorithms in games development but often have nothing to do with each other. The overall aim of this project is to see if these two very different algorithms can be integrated to produce a unique mechanic that could potentially be used in games. This aim is divided up into four objectives with the first two involving the implementation of a pathfinding and lighting algorithm, the third involving the implementation of a basic link between the co-ordinates of the light and the pathfinding algorithm and the fourth being the implementation of additional lighting variables such as attenuation to calculate whether or not the AI can travel through the nodes based on a set sensitivity variable.

The project has been carried out in the form of an experiment due to need to manipulate independent variables and monitor the outputs for trends. The main results from this are the light intensity values for each node calculated by finding the cosine of the angle between two nodes to the power of the attenuation variable. These values are compared against the sensitivity of the AI so that any node with a light intensity greater than the sensitivity cannot be travelled through by the pathfinding algorithm. The results can be seen when the program is run, the console window should be kept visible as it displays the light intensity values for each node and provides validation checking. The results change depending on the variable being altered, these variables include the spotlight position, attenuation value and AI light sensitivity although this variable only has an effect on the AI, not the graph.

Table of Contents

Abstract.....	4
Introduction	6
Research & Review of Literature	8
Introduction	8
Pathfinding.....	9
Dijkstra	9
A*	11
Lighting.....	13
Conclusion.....	15
Research Methodology	17
Design.....	20
Implementation	24
Evaluation	35
Conclusion.....	39
Reflection	42
References	44

Introduction

The aim of this project is to investigate the possibility of implementing light calculations into a shortest-path pathfinding algorithm to increase path complexity. The aim can be achieved by accomplishing four objectives listed below. These objectives will be reviewed in the evaluation to determine whether or not the experiment was a success:

- The pathfinding algorithm can calculate the shortest path between two points
- The lighting variables can be adjusted while the program is running
- The position of the lighting affects the shortest-path calculation
- The intensity of the lighting affects the shortest-path calculation

The first objective is the implementation of the pathfinding algorithm, this objective must be met. The second objective requires that the light be adjustable while the program is running, this includes position as a minimum with intensity and angle being integrated for the fourth objective. The third objective is whether any attributes of the light source such as its position have any effect on the shortest-path, this means that if the light is directly above a node it will be blocked, making traversal through it impossible. This objective must also be met as this is the most basic level of implementing a light-sensitive AI. The final objective requires that any node with an attenuation value above the light-sensitivity value be avoided by the AI and any value below being safe to traverse.

The main purpose of this experiment is to see whether or not light can be successfully implemented into a shortest-path pathfinding algorithm to increase its complexity, the

results of the experiment can potentially be used in games that require light-sensitivity. This would prove most useful in stealth and horror genre games.

The requirements to complete this experiment include the coding of a pathfinding algorithm and a graphical user interface with which to implement the lighting. Should the integration between the pathfinding and lighting prove successful various tests will need to be performed to see if the results are noticeable. This could include adjusting the lighting or testing different maps for the pathfinding algorithm to explore.

The following chapters include a review of the literature used in researching this experiment, an explanation of the research methodology used to create it, a documentation on the design and implementation including results followed by an evaluation and conclusion. Finally there is a reflection which is a look into the work done, explaining what went well and possible advancements that were not included in the final experiment.

Research & Review of Literature

Introduction

Artificial Intelligence encompasses a wide array of implementations, this can range from simple pathfinding techniques to programs capable of learning. In particular to gaming; pathfinding is often used with other techniques in order to increase the complexity and originality of the game. Lighting also plays a prominent role in games, namely 3D games that use lighting to set more realistic or appealing visuals. However there are few games that integrate the two together. One technique that does combine the two but is often overlooked is light-sensitivity, which is predominantly found in horror and stealth games. These genres particularly use light as a deterrent or to adjust the enemy AI's ability to detect the player respectively. While there are many games that use light-sensitivity; there is little documentation on it, with most articles on light-sensitive pathfinding being based around programming robotics rather than games AI.

This literature review will focus on two main points, pathfinding algorithms and lighting, the pathfinding algorithm will be the very base of the program as it is a necessity to the experiment, while the lighting is also important it could prove to be too problematic to integrate the two. In this situation the lighting will be replaced in favour of mathematical calculations that will simulate how the light would work.

While evidence of this project can be seen in some games there is no documentation for it, this means that there is no data for comparison which lead to the project becoming an

experiment. However, the project can be divided up into sections, namely the pathfinding and the lighting which will be the focus of this literature review.

Pathfinding

The pathfinding algorithm is the base of this experiment, the AI needs to find the shortest path from its starting point to the goal. In order to get the best results each algorithm must be evaluated for its pros and cons, after this has been done the algorithms will be compared to see which is most suited to be used in this experiment. Possible pathfinding algorithms include A* and Dijkstra, with the difference between the two being that A* uses a heuristic function to estimate the cost of the path.

Dijkstra

A good paper that gave a basic understanding of Dijkstra's algorithm was written by the authors N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma and N. Nosovic [1]. The paper covers the implementation of Dijkstra's algorithm in parallel programming. Even though this information is not relevant to the experiment the paper gave a simple step-by-step explanation of how Dijkstra's algorithm should be laid out. An image was provided of how the algorithm searches nodes which was relatively clear but was small enough that some of the labels were hard to read. The rest of this paper went on with explanations and results on parallelisation which ended with Dijkstra's algorithm being slower than expected which the authors suspect is due to inefficient coding. While no truly useful information could be extracted from this paper it did help with understanding some of the terms in the more complex papers.

A similar yet more relevant paper written by D. Fan and P. Shi [2] focused on Dijkstra's algorithm in route planning. The paper starts by explaining some of the disadvantages of the algorithm including high time complexity and large storage space which are points to consider when implementing the pathfinding algorithm for the experiment. The paper explains various methods that attempt to improve the algorithm including the usage of adjacency and memory matrices to store data; however some of the methods such as the adjacency matrix store a series of 0s and infinities which has a reverse effect on speed and storage. The memory matrix method uses linked lists and appears to be a better alternative due to only the non-infinity elements being stored. Even though the paper is easy to read and understand there are a few errors in the text that may cause confusion, though this may be due to English not being the authors' first language. Also some variables are used that have not been explained leading to more confusion when following the steps.

A fair amount of diagrams have been illustrated with most of them being very simple but easy to understand, however, the graph traversal diagrams are very small which makes it hard to understand what the authors were trying to show. Thankfully they have been labelled and it can be seen that the pathfinding algorithm needed to navigate fewer nodes to find its destination. Towards the end of the paper a table comparing the standard Dijkstra's algorithm with an altered version was shown, the results showed a vast improvement in calculation speed and limited the number of nodes traversed.

Consequently it seems that the algorithm restricts the search area and will not always find the shortest path, leading to the authors themselves calling the algorithm "defective". While

the information seemed useful at first, the fact that the results led to some problems finding the shortest path made the idea of using a similar algorithm in the experiment less appealing.

A*

The first article by J. Hu, W. gen Wan, and X. Yu [3] explains the basics of A* pathfinding, this paper is based in Unity and focuses on using pathfinding for Real-Time Strategy games, the pathfinding for this sort of game requires recalculation of the pathfinding algorithm every time the player or enemy moves which relates to how the algorithm would need to adapt in this experiment every time the lighting is altered. The information included in this article is very basic and includes little more than an explanation on heuristics and the Manhattan method, it provides little pseudo-code but was useful as a starting point with various images of how it works in theory along with a finished product. Similar to the first paper read for Dijkstra's algorithm, this paper seems like a good start for understanding how A* pathfinding works, however there is little information that can be implemented into the experiment without researching further.

The next paper by V. Sharma and R. Kumar[4] explains the need for finding the optimal path in graphs, it goes into far deeper explanation than the previous paper and goes step by step through the mathematics of A* pathfinding with special attention to the heuristic value. Later on the authors give several theoretical situations in which A* is compared with another algorithm in order to prove whether A* will return an optimal value.

The logic appears sound and is written in great detail no actual proof has been provided or any sign of implementation into a program. The pathfinding algorithms referenced are hypothetical, with no reference to actual pathfinding algorithms until the conclusion. While the previous paper by J. Hu, W. gen Wan, and X. Yu [3] may seem too simple this paper seems far more complex and the text is harder to follow, with many variables from previous pages being referenced which requires backtracking through the article. At the end of this paper the authors simplify the text in the last few paragraphs, with their final conclusion being “The selection of h , therefore, permits one to choose a desirable compromise between admissibility, heuristic effectiveness, and computational efficiency”[5]. The letter h refers to the heuristic value while the quote states the usefulness of using a heuristic value in a shortest-path pathfinding algorithm in balancing out speed and precision as long as the heuristic function is admissible (Cost is not overestimated). The information from this paper may be too complex to start with, it may be beneficial to return to this paper after some progress has been made implementing a pathfinding algorithm for the experiment. This may provide a better understanding and should help in improving how the algorithm works.

A well-presented paper was published by Z. Xu and M. Van Doren [6]. It explains the A* algorithm in relative depth and is written in the context of map traversal in a game, fairly similar to the light-sensitivity experiment. The paper is well-structured and includes a literature review although only a handful of references are used throughout the paper. Like all previous A* algorithm papers it features the formula for A* using a heuristic value “ $f(x) = g(x) + h(x)$ ” [6]. The paper also explained the necessity for lists in pathfinding in order to

store the nodes depending on whether they have been visited, not been visited or are part of the path to the goal, this information is useful for when it comes to implementing the pathfinding algorithm should A* be chosen.

The paper then goes on to show a game using pathfinding implemented in Java, the path is dynamically generated meaning that should something obstruct the shortest path then the route will be recalculated. This is very relevant to the experiment which, as stated earlier, will need to recalculate the route every time the lighting is adjusted. Various images are shown of the game and how the path is generated and adjusted when the current shortest path is gone, followed by pseudo-code showing how nodes are checked and added to the shortest path depending on the cost. This could prove a useful reference for implementing a basic A* pathfinding algorithm as a starting point. Finally the paper illustrates some other features of the game that require recalculating the shortest path but they are not relevant to the experiment.

Lighting

Very little documentation was provided on lighting itself, with no articles having much relevance to the experiment. The experiment will involve the use of OpenGL as a graphics library, so relevant information would include the implementation of OpenGL lighting such as point lights or spot lights along with any information on light adjustment such as attenuation (or exponentiation) which is the rate at which the light intensity is lowered from the centre, giving the appearance of realistic lighting which grows dimmer the further away from the centre it is.

One of the few papers relevant to the lighting aspect of this experiment was written by D. Harris [7], his paper is focused on the exponentiation unit for lighting in OpenGL. There were a few problems with the structure of the paper that made it hard to follow, the spacing between new lines and text was small which made it hard to read and overbearing. While it is expected that the reader have some prior knowledge to the terms and variables used in this type of paper there was no explanation to the values used with only a simple diagram to give a poor explanation of how the variables relate to each other, making it hard to use any techniques referenced. The introduction was also one of the largest parts of the paper but did provide a small bit of information into the calculation of the light intensity.

D. Harris wrote "Both specular lighting and directed spotlight calculations involve raising the cosine of an angle to a power to determine the light intensity reaching the viewer" [7]. Once the paper had reached its 'Error Analysis' section the formatting changed, from here a small amount of pseudo-code was provided that checked the values of the exponent to make sure they weren't too small to make a difference or to convert them to the right format. The error checking may prove useful in the experiment to avoid any unexpected or unwanted values being output.

From here small paragraphs were used, each roughly explaining the calculations involved in the error checking. Some of the calculations were so long that the author had to adjust the text size, making it hard to read the formulae. It also seems that each formula was referenced from a different paper, unfortunately most of these consisted of look-up table calculations which had no real relevance to the experiment.

Overall this paper seems very informative and may prove useful to someone with a similar level of knowledge as the author but aside from a few basic calculations that may prove useful for the experiment; the paper seems too complex for what is required.

Another paper published by D. Harris [8] focuses on using the OpenGL pipeline, the text is simpler than the previous article but still has the same problems as the previous paper, with unexplained variables and formulae. However some of the formulae are the same as those in the previous paper which may suggest calculating exponent. Without further research into the calculations for OpenGL it is impractical to try to implement any of the equations into the experiment. Instead it may be better to return to these papers when a better understanding of OpenGL lighting is achieved to see if they can be of use.

Conclusion

From the research gathered here it is clear that there is a lot more documentation for pathfinding than there is for lighting in relevance to the light-sensitivity experiment. In addition to this from what was found in the pathfinding papers A* has been chosen as the pathfinding algorithm for the experiment. This is because the advantages from using A* listed in the papers seem to fit better with the experiment which requires fast calculations. In addition to this one of the papers that used Dijkstra produced negative results although this was due to testing alternative methods for optimisation.

There was very little information useful for the experiment at this present time due to the complex calculations used with no reference to what they are. It may prove useful to reference back to these papers later on when a better understanding of coding with OpenGL has been achieved.

Research Methodology

The main research methodology used in the experiment was the experimental method, this is because the experiment relies on manipulating variables in order to identify how the AI reacts and to see if any trends exist. The experimental method has the advantage of being able to manipulate the variables which allows full control of how the experiment responds, this makes trends and outputs easier to identify.

The main problem with the experimental method in this situation is that there is the possibility of data being interpreted incorrectly due to interpreting visual differences by eye (human error). This would cause serious problems as the results would be invalid. In order to ensure that this doesn't happen the values of the attenuation are printed onto the console. These values can be compared against the light-sensitivity variable of the AI, if the attenuation value is higher than the light-sensitivity value then the pathfinding algorithm should not go through it.

Another disadvantage that stems from human error is if the experiment has been conducted incorrectly, in this scenario the outputs may seem correct but there could be an underlying fault that may cause problems. The experiment is evaluated based on whether the AI adjusts its path when the current shortest path is intercepted by a light-source above the current light-sensitivity value.

At the end of the experiment it was expected that there would be a working program that will take light levels into account when calculating the shortest-path. A threshold should be

set which will act as a limit to what light levels the AI can move in, anything above this should make the AI avoid the route at all costs. The lighting and path should be adjustable within the program, this means that the program should not be required to be closed in order for the changes to take effect. This is to closer simulate what would be expected of the program in a commercial product.

There are no particular requirements for hardware in this experiment as long as the hardware can support the software required. The software used for this experiment includes Visual Studio 2013 as the development interface, along with this OpenGL will be a crucial library in this experiment in order to implement the lighting and graphics.

The experiment will focus on a few adjustable inputs that will affect what path the AI will take. These inputs mainly include the lighting variables such as position, radius and intensity. With these the lighting can be scaled and moved about inside the program which should have a visible effect on the shortest-path. Another variable that can be potentially changed is the graph which the AI will traverse, this can add to the complexity of the path further and should result in larger adjustments should the shortest path be blocked by the lighting.

The experiment should provide evidence of its reliability when run, the results should be clear enough to see on the graphical application and if not the values of each node in the graph should be printed onto the console. These values can be compared to the light sensitivity of the AI, the path should not go through any nodes with a light level greater than its light sensitivity.

The validity of the experiment can be shown by adjusting the lighting variables, when one of these variables is altered enough it should have a visible effect on the pathfinding algorithm, these effects should be persistent and trends in how the path is recalculated should be noticeable when adjusting the light.

The experiment starts by testing to see if the pathfinding algorithm calculates the shortest path (which may be different depending on which graph is used). A small light should be noticeable in the middle which may have already have an effect if the start and end points are either side of it. The next test involves checking to see if the light can be altered while the program is running. This can be done using the keyboard keys to alter position, angle and intensity. Next the light is moved over a node that is in the shortest path to see if the path is recalculated. If it is that means the light has had an effect on the pathfinding algorithm.

The angle of the light is then altered, due to the attenuation starting low increasing the angle enough should cause surrounding nodes to also become unsuitable for the shortest path. The final test involves increasing the attenuation which will lower the light intensity, this should cause the nodes at the edge of the light to become eligible for the shortest path again. This experiment is verified by eye but certain variables should be printed to console as definite proof of the experiment working or not.

Design

As there was no documentation on anything similar to this experiment it was difficult to find out what was needed to advance the experiment, in order to counteract this a development methodology similar to prototyping for solo programming was used, this methodology was developed by A. Hollar and is called the Cowboy Methodology [9]. Throughout the development a list of required features was written based on the objectives stated in the introduction.

The first objective required the implementation of a pathfinding algorithm. A* was chosen as the algorithm due to the amount and quality of information found in the literature review, along with this there was also the fact that A* can perform faster than Dijkstra's algorithm if done correctly while only requiring the addition of a heuristic function.

The papers from the literature review and pseudo-code from various online articles were reviewed in order to find the common methods used in the algorithm. One of the major problems when designing the pathfinding algorithm was the fact that each example had a different kind of output, some examples would end up with an array of values while others would simply output the x and y of each node travelled, this will be discussed in further detail in the implementation section.

Eventually an example of actual was found that was suited for the experiment.

Unfortunately the algorithm was structured in a way that made changes difficult, this algorithm was used as the base of the program and was adapted and optimised slightly to

better suit the experiment. A reference to the original code can be found inside the source code of the experiment.

After the pathfinding algorithm was implemented and tested the next step was to create the lighting, OpenGL was already chosen as the graphics library at the beginning of the experiment and it was decided that the rendering pipeline would be used to create the lighting due to ease of implementation. This is because the main focus is the pathfinding algorithm.

A spotlight was chosen as the lighting method for this experiment due to its suitability, it is a directional light meaning that it can be pointed in a specific direction and more importantly only lights up objects within the cone angle. After researching a bit on the rendering pipeline a function called `glLight` was found. This function had parameters that met with the desired alterations that were planned for the lighting. The alterations include the adjustments of the light angle, positioning and intensity. These are the independent variables for this experiment which means they will be altered manually. Another independent variable that is necessary for the experiment is the one that sets the sensitivity level for the AI, this will be compared against the light intensity of each node but can also be controlled by the user to adjust what levels of light the pathfinding algorithm can handle.

The independent variables can be seen in the following table:

spotAngle	Half the angle of the spotlight
lightPos	Position of the light
spotIntensity	Controls light intensity
aiSensitivity	Sets what light intensity the pathfinding algorithm can handle

If this experiment has independent variables then it will also need dependent variables. The values of these variables will be altered depending on the values of the independent variables, with a trend that should be noticeable.

attenMap	Stores the intensity values for each node, value altered depending on spotIntensity value
----------	---

There is only one dependent variable for this experiment, this attenuation map stores the attenuation value of each node which is checked against the aiSensitivity to see if the node is too light to travel through.

Along with these variables the standard functions for an OpenGL project would need to be created as well. These include a function to set up the lighting, a function to draw the scene and a function to resize the window should the user adjust it. Another function that is not typically included was the keyboard input function, at the start it was planned to control the

light using the mouse but due to the amount of independent variables that needed to be manipulated it was simpler to integrate everything into keyboard controls.

While the first two objectives were tested before moving onto the next phase the third objective was bypassed due to expectations that any progress made during this iteration would have to be changed to implement the fourth objective. At this stage the `spotAngle` and `spotIntensity` had a visible effect on the lighting but played no role in the pathfinding calculations.

When the implementation for the angle and intensity occurred there were problems calculating whether nearby nodes were within the angle of lighting, this will be explained in further detail in the implementation section. This lead to having to removing `spotAngle` as an independent variable and changing it to a constant. This does not cause any problems with the objective as the most important variable to test light sensitivity is `spotIntensity`.

Implementation

As stated in the design section the program was developed using a method similar to prototyping, at each design phase a scope was set then developed. The process would then repeat with a larger scope being set. The first implementation involved the creation of the A* pathfinding algorithm. There were a few issues when implementing the algorithm, one of the main problems was that all the sources that were being used for reference were manipulating the data in a way that wasn't useful to the experiment.

Most of the example algorithms simply output the x and y co-ordinates of each node travelled through in a list while the experiment needed the values printed out to an array. One web article did contain the required output but was structured in a way that left little possibility for adapting the code, originally there were a few memory leaks and there was a delay when the program was calculating the path but these problems have been fixed in the experiment. The article that the pathfinding algorithm is based off has been referenced in the source code. The output of the pathfinding algorithm can be seen in Figure 1.

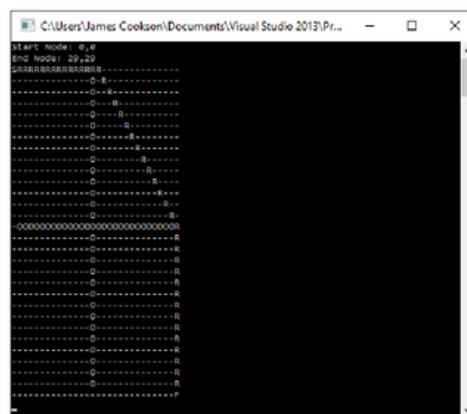


Figure 1

The next implementation phase was the lighting, due to the use of the fixed rendering pipeline along with referencing back to previous assignments for the structure the lighting was implemented with few problems. As the nodes from the pathfinding algorithm had not been implemented yet a random set of cubes equidistant from each other were created. The spotlight was added with a default angle of 10, this meant that only the node the light was above would be lit as can be seen in Figure 2.

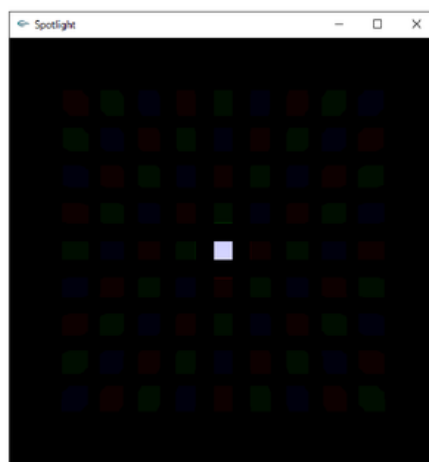


Figure 2

The independent variables were already set as the values were required to create the spotlight. This made manipulating the values easy, especially when the keyboard inputs were implemented. By pressing the U or I keys the user can decrease or increase the angle of the light respectively, this can be seen in Figure 3.

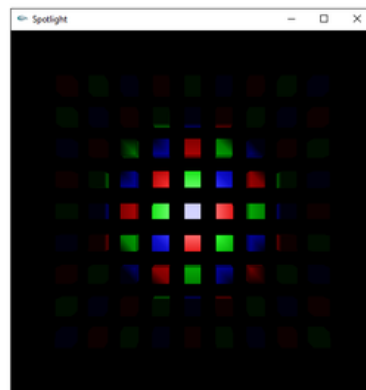


Figure 3

By using the J key the user can manipulate the light intensity in order to give the appearance of the light fading out as it reaches outwards. Alternatively the user can increase the intensity by using the K key which will make the light much brighter, with the light simply cutting off when it reaches past the angle of the spotlight. This can be seen in Figure 4 below.

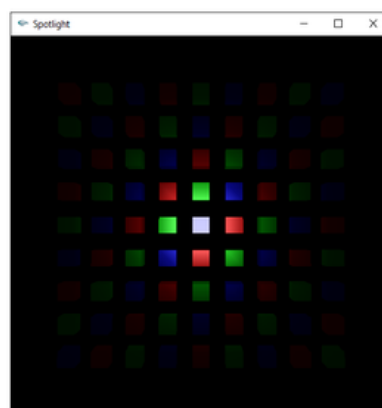


Figure 4

After implementing both the lighting and the pathfinding algorithm the next step was to merge the two together, the for loop that created the random cubes was replaced by a loop that displayed the cubes in a different colour depending on the value of the pathfinding array seen in Figure 1. The result can be seen in Figure 5, with blue being the start, purple being the path, red being open spaces, green being obstacles and light blue being the finish.

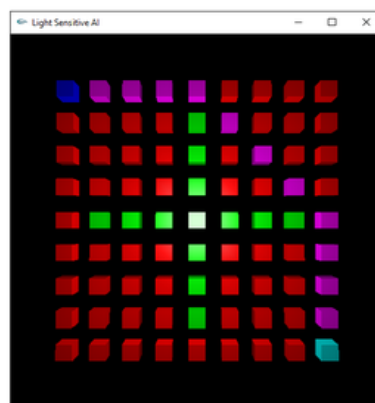


Figure 5

The movement of the spotlight has not been demonstrated yet, by using the W, A, S and D keys the user can move up, left, down and right respectively. Figure 6 shows the light source moved to $x = -3$ and $y = -3$, the shortest path has already been adjusted but this is due to the new path being the same length as the old path. Figures 3, 4 and 6 meet the requirements of the second objective which requires the ability to alter the lighting while the program is running.

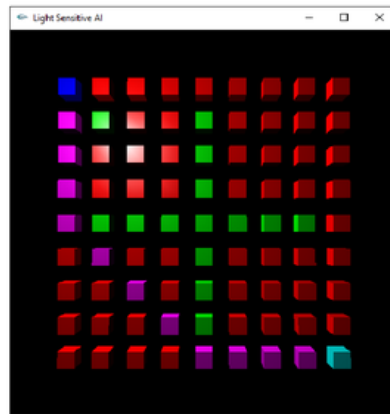


Figure 6

Before the implementation of attenuation the location of the light was calculated on the pathfinding algorithm through a `for()` loop. If the x and z co-ordinates in the graph were equal to the x and z co-ordinates of the light then the node could not be traversed. This code was removed when the attenuation calculations were integrated due to redundancy.

Figure 7 shows the light source again on $x = -3$, $z = -3$ but with the light reduced, the next step will move the light to $x = -4$ and should change the path back to its original route as seen in Figure 8.

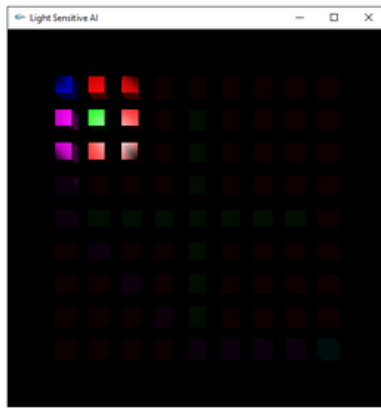


Figure 7

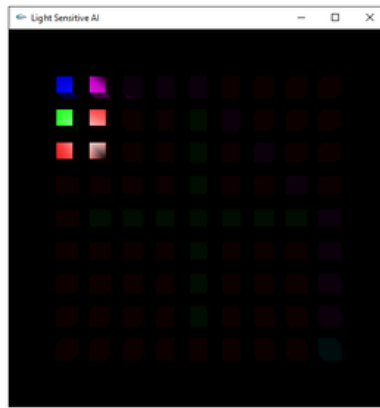


Figure 8

During the implementation for the fourth objective a problem was encountered due to there being no identifiable trend in the angle from the spotlight centre to the nodes. The angle between two nodes is calculated and then used in the calculation of the attenuation for each node, the code for the calculation of the angle between two nodes can be seen below. While the results of the calculations seem to be correct, had no connection to the `spotAngle` variable. This means that the results being displayed on the screen and what would be used in the calculation of the path would be completely different, producing incorrect results.

```

a = { 0, -1.0, 0 };
b = { iSize, -1.0, jSize };

//Find length of both vectors
float aLength = sqrt(a.x*a.x + a.y*a.y + a.z*a.z);
float bLength = sqrt(b.x*b.x + b.y*b.y + b.z*b.z);

float dot_product = (a.x*b.x) + (a.y*b.y) + (a.z*b.z); //D
float fullLength = aLength*bLength; //Length of a * b
float theta = acos(dot_product / fullLength); //Angle betw
float thetaDegrees = (theta * 180) / PI; //Convert to degr

mapAngle[i][j] = thetaDegrees;

```

This led to the design being altered to remove the spotAngle variable as an independent variable and instead make it a constant at max range. This also dealt with the problem of nodes not in the light source being too hard to see on screen. The previously calculated angle is then used to work out the attenuation as seen below.

```

lightCos = cos(mapAngle[i][j] * PI / 180.0);
lightIntensity = pow(lightCos, spotIntensity);

if (lightIntensity < 0) lightIntensity = 0;
attenMap[j][i] = lightIntensity;
if (attenMap[j][i] >= aiSensitivity && graph[j][i] == 0)
{
    graph[j][i] = 1;
}

```

This is done for every node in the array, the values are then output to the console as a check to validate whether or not the path has been calculated correctly, an example of the output can be seen in Figure 9.

```
C:\Users\James Cookson\Documents\Visual Studio 2013\Pr...
Press the u/i keys to increase/decrease the AI sensitivity
Press the wasd keys to move the spotlight.
Press j/k to decrease/increase the spotlight's intensity.

0.030 0.038 0.048 0.056 0.059 0.056 0.048 0.038 0.030
0.038 0.053 0.071 0.091 0.100 0.091 0.071 0.053 0.038
0.048 0.071 0.111 0.167 0.200 0.167 0.111 0.071 0.048
0.056 0.091 0.167 0.333 0.500 0.333 0.167 0.091 0.056
0.059 0.100 0.200 0.500 1.000 0.500 0.200 0.100 0.059
0.056 0.091 0.167 0.333 0.500 0.333 0.167 0.091 0.056
0.048 0.071 0.111 0.167 0.200 0.167 0.111 0.071 0.048
0.038 0.053 0.071 0.091 0.100 0.091 0.071 0.053 0.038
0.030 0.038 0.048 0.056 0.059 0.056 0.048 0.038 0.030

Start Node: 0,0
End Node: 8,8
Attenuation: 2.000
AI Sensitivity 0.200
```

Figure 9

The value of AI Sensitivity in this example is 0.2, this means that any above 0.2 cannot be travelled through by the pathfinding algorithm. The only values above 0.2 are the nodes surrounding the node directly below the spotlight as seen in Figure 9, this should be reflected in Figure 10.

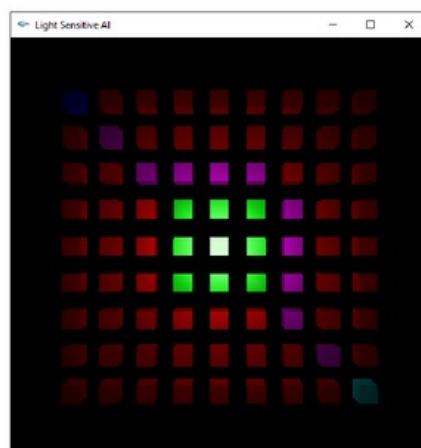


Figure 10

The graph was made blank in order to properly demonstrate the effects of the light sensitivity. Decreasing the attenuation value should have an effect on the values printed onto the console, this in turn should visibly affect the pathfinding algorithm. The effects from changing the attenuation from 2.0 to 1.3 on the intensity calculations can be seen in Figure 11, while effects on the pathfinding algorithm can be seen in Figure 12.

```
0.103 0.120 0.138 0.153 0.159 0.153 0.138 0.120 0.103  
0.120 0.148 0.180 0.210 0.224 0.210 0.180 0.148 0.120  
0.138 0.180 0.240 0.312 0.351 0.312 0.240 0.180 0.138  
0.153 0.210 0.312 0.490 0.637 0.490 0.312 0.210 0.153  
0.159 0.224 0.351 0.637 1.000 0.637 0.351 0.224 0.159  
0.153 0.210 0.312 0.490 0.637 0.490 0.312 0.210 0.153  
0.138 0.180 0.240 0.312 0.351 0.312 0.240 0.180 0.138  
0.120 0.148 0.180 0.210 0.224 0.210 0.180 0.148 0.120  
0.103 0.120 0.138 0.153 0.159 0.153 0.138 0.120 0.103  
  
Start Node: 0,0  
End Node: 0,8  
Attenuation: 1.300  
AI Sensitivity 0.200
```

Figure 11

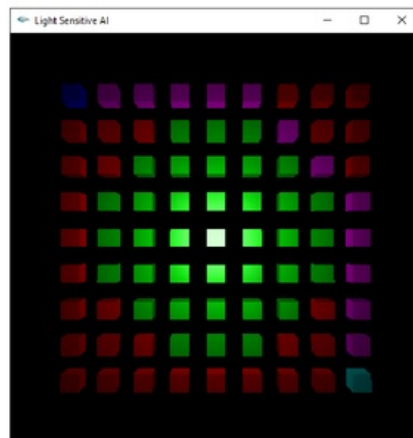


Figure 12

The demonstration of the program in action in Figures 9-12 show that the light intensity does affect the pathfinding algorithm to a noticeable degree. While the results are clear from the graphic representation the math can be validated by checking the `aiSensitivity` value against the light intensity of each node displayed in the console.

While the end result still meets the objectives, one of the initial expectations was to limit the attenuation calculations to nodes within the `spotAngle`. No solutions were found to implement it and the reason may possibly be due to using the fixed function pipeline to implement the lighting. This limited the control over the calculations for displaying the spotlight which, while correct, did not match with the calculations used in the rest of the code.

Initially it was planned that the attenuation calculations would be limited to nodes within the `spotAngle`. This could not be implemented as the angle values in `spotAngle` shared no correlation to the angle values between nodes.

The problem was bypassed by discarding the spotlight angle as an adjustable variable and using the node angle calculations to work out the intensity. This did not prove too much of an issue as the spotlight angle was not strictly necessary for simulating realistic lighting. This could instead be handled by adjusting the attenuation variable. However this means a cut-off point could not be implemented which would have given more control on adjusting light intensity values.

The reason for why the angle values did not match could be due to using the fixed function pipeline, both calculations seem to produce correct results but as the spotlight is created using the pipeline there is a limit to how the spotlight can be controlled.

A side effect of setting the angle as a constant was that the entire graph was visible, initially there was an issue where nodes outside the spotlight angle were too dark to see. A solution was planned involving the implementation of another light that would make the graph lighter. While this would have made the spotlight harder to see it was also planned that the light could be toggled on and off using a keyboard key.

Evaluation

In order to successfully determine whether the experiment was a success or not it needs to be evaluated against the aim. In order to see if the aim was achieved each objective needs to be looked into, the objectives are:

- The pathfinding algorithm can calculate the shortest path between two points
- The lighting variables can be adjusted during runtime
- The position of the lighting affects the shortest-path calculation
- The intensity of the lighting affects the shortest-path calculation

The first objective was met by implementing a pre-existing A* pathfinding algorithm and adapting it to suit the experiment, the reference to the original code can be found in the source code. The second objective involved the implementation of a light source that could be controlled during runtime. This objective was achieved with little trouble but the result ended up slightly different to the original idea. Initially the goal was to have the centre of the light be determined by the position of the mouse. During development keyboard keys were used as a basic method of variable control until the code for mouse-tracking was implemented. This idea was abandoned due to the number of variables that needed to be controlled during runtime, the user would have ended up having to use both the mouse and keyboard to control the variables which seemed to overcomplicate things for the user.

The third objective was the first objective that produced any actual results, the first two objectives were simply there to get to this stage. Before implementing angle and intensity

equations there needed to be a test to see if the lighting could be used to have an effect on the shortest-path, the simplest way to do this was to use the position of the light. If the light was over a node then that node could not be travelled through, as stated in the implementation the code that implemented this was removed as it became redundant when the attenuation calculations were integrated.

The final objective was the actual test to see if light values could affect pathfinding. This meant using the angle and attenuation values in an equation to calculate the light intensity for each node, this value would then be compared to the level of light the AI can travel though and if the value was higher the AI would not be able to travel through the node. As was stated in the implementation a problem was encountered when calculating the angle as the spotlight angle handled by the fixed function pipeline produced different results to the angle calculations between two nodes.

This problem was bypassed by discarding the spotlight angle as an adjustable variable and accurate results were still produced without it at the cost of the extent to which the spotlight could be controlled. The light intensity values for each node were calculated using the angle between each node and the current attenuation value, these values were then checked against the light sensitivity value of the AI. Values higher than the light sensitivity could not be passed through and were displayed the same colour as obstacles in the graphics interface. In order to check whether or not the calculations were correct the light intensity values were printed to console which helps to prove the validity of the results.

As a reminder the aim was to investigate the possibility of implementing light calculations into a shortest-path pathfinding algorithm to increase path complexity can be increased. As all four objectives have been achieved it has been proven that light calculations can be implemented into a pathfinding algorithm and values such as attenuation can be implemented in order to control how light the path can be before the algorithm refuses to travel through it.

In order to interpret if the results give a realistic representation of the path adjusting to dark areas the user needs to judge by eye. While this may cause the user to question the validity of the result the program gives a clear representation of the lighting and a trend can be visually perceived when adjusting the position and attenuation value of the light along with the sensitivity of the AI.

If the experiment were to be conducted again there are a few additional features that would be considered. One of the features is the capability to zoom in and out on the y axis. While this is a simple feature it would allow the program to display larger graphs. The program supports square graphs of different sizes but is limited due to the position of the camera.

Another potential feature is the implementation of more lights, this was considered as a possible extension to the experiment but was not part of the original specification. Below is a code snippet of the spotlight implementation.

```
//Light properties.  
glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);  
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDifAndSpec);  
glLightfv(GL_LIGHT0, GL_SPECULAR, lightDifAndSpec);  
  
glEnable(GL_LIGHT0); //Enable particular light source
```

The spotlight is GL_LIGHT0, more lights could be implemented by using GL_LIGHT1 up to 8.

The calculation for light intensity would not need must adjustment as it would be checked for each light with the higher value being recorded into the array. When the position of a light is changed the program would need to know which light is being moved, this could be done by implementing a key that would toggle to the next light.

Shaders would be considered as an alternative to the fixed function pipeline, while this would increase the complexity of integrating lighting it would give a lot more control which may have circumvented the problem with the spotlight angle during implementation.

Conclusion

While the objectives have been met there was very little that was extracted from the literature review. Due to the limited documentation on lighting there was little knowledge that proved useful. In addition to this the reviewed articles were written under the assumption that the reader has a good knowledge of implementing lighting in OpenGL. This meant relying on previous experience and online documentation which was one of the reasons for using the fixed function pipeline.

There was a great deal more on pathfinding algorithms with a better layout of formulae used and images relating to the results of using those formulae. Again these papers were not referenced much for the development of the experiment as the example programs were oriented towards the research the authors were carrying out.

Overall the literature review provided little information that could prove useful in the experiment, this was predicted however as initial searches showed no similar research being carried out.

The design was carried out using a methodology similar to prototyping for solo programmers, this proved appropriate considering the little information extracted from the literature review. The scope of each iteration was based on each objective, with the first iteration producing the pathfinding algorithm and the second producing the lighting with controls and so on.

This allowed a good idea of what needed to be implemented in each iteration although there was a fair amount of trial and error to figure out exactly what needed to be included in order to get the program to work as predicted. When it came to integrating the lighting into the pathfinding the independent variables were already implemented which made it easier to integrate the new code. As the results of the experiment are mostly based on whether or not the visuals correctly represent the adjustments made to the independent variables it was important to include screen captures of the program working in various states. These images also show the step-by-step progression of the program from the two prototype programs for pathfinding and lighting to their integration into the main program.

The evaluation didn't contribute to meeting the objectives but does look back at the aims and objectives stated in the introduction and explains what was done to meet them. It also includes some of the additional features that were being considered but were not implemented.

One of the purposes of this experiment was to demonstrate that two separate algorithms, while both are often in games, rarely have anything to do with each other. Proof of this can be seen from the lack of research done in integrating light and pathfinding. Integrating these two very different algorithms together creates an interesting and unique mechanic which could make a game that includes it stand out compared to similar games.

The program produced can be altered slightly to completely change the behaviour of the AI. For example the intensity check can be inverted so that the pathfinding algorithm will only travel through light areas. The end goal of the shortest path can also be changed into a

dependent variable that is equal to the lightest or darkest node in the graph, this will make the AI constantly chase after the node with the highest or lowest value. These are very different behaviours from the current program but only require some small changes to the code.

Reflection

During this project I have learnt many things that I believe will help me in the future both academically and in the job industry.

As this is my first dissertation I am not used to writing such long reports, writing the dissertation has led me to split up my work better so that it is easier to read. One example of this is moving to a new paragraph when moving onto the next topic, while it may seem like a trivial thing it is harder to follow text when few paragraphs are used and new topics are introduced on the same line as the previous topic.

Another improvement is my increase in vocabulary. Through the writing of this dissertation I have found that I tend to repeat the same words multiple times. This causes issues with the flow of the text which led me to look up alternatives to the commonly used words in a thesaurus.

My styling of documents has also changed, for this dissertation I learnt how to format a few things in Word such as the header styles so that a format could be applied instantly when a new chapter was started. By using the header styles the table of contents would recognise that the text was a title and would display it automatically.

The project has greatly affected my time-keeping and planning skills. While I am ashamed to say I used to leave things up until the last minute the sheer scale of the write-up led me to plan out a schedule to follow each week, with a time set for programming the experiment and writing out the dissertation. Towards the end of the project it became more clustered

due to trying to finish things off in time but had I not prepared the plan I most likely would have been worse off.

My problem solving skills have definitely improved as a result of this project, while in previous programming assignments there were documentations or similar examples of code that could be used for reference. This was not available for the integration of the lighting and pathfinding which required constant trial and error to get the program to work as desired. At first one problem could cause a couple of hours of wondering what was wrong, towards the end of writing the program I could better identify what went wrong if the debug window did not provide enough information and fix most of the problems I was having in a few minutes.

References

- [1] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma, and N. Nosovic, "Dijkstra's shortest path algorithm serial and parallel execution performance analysis," in *MIPRO 2012 Proceedings of the 35th International Convention*, 2012, pp. 1811–1815.
- [2] D. F. D. Fan and P. S. P. Shi, "Improvement of Dijkstra's algorithm and its application in route planning," *Seventh Int. Conf. Fuzzy Syst. Knowl. Discov. (FSKD)*, 2010, vol. 4, no. Fskd, pp. 1901–1904, 2010.
- [3] J. Hu, W. gen Wan, and X. Yu, "A pathfinding algorithm in real-time strategy game based on Unity3D," *2012 Int. Conf. Audio, Lang. Image Process.*, pp. 1159–1162, 2012.
- [4] V. Sharma and R. Kumar, "Direction Based Heuristic for Pathfinding in Video Games," pp. 1551–1557, 2015.
- [5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.
- [6] Z. Xu and M. Van Doren, "A Museum Visitors Guide with the A* Pathfinding Algorithm," pp. 62–66, 2011.

- [7] D. Harris, "An exponentiation unit for an openGL lighting engine," *IEEE Trans. Comput.*, vol. 53, no. 3, pp. 251–258, 2004.
- [8] D. Harris, "A Powering Unit for an OpenGL Lighting Engine," *Evaluation*, pp. 1641–1645, 2001.
- [9] A. B. Hollar, "Cowboy: An Agile Programming Methodology for a Solo Programmer," no. December, 2006.