2010 International Conference on Business and Economics Research
vol.1 (2011) © (2011) IACSIT Press, Kuala Lumpur, Malaysia

Autonomous Object Movement in Modeling Bees Foraging

Mustafa Muwafak Theab[1], Yuhanis Yusof[2]

College of Arts and Sciences
Universiti Utara Malaysia
Kedah, Malaysia
1new.technology@hotmail.com, 2yuhanis@uum.edu.my

***Abstract*—This paper presents an algorithm inspired by Bresenham's algorithm to move an object from one location to another. This algorithm uses the slope calculation for different location and direction to move an object through the desired path. Such an approach has extended the use of existing Bresenham algorithm into a better use as direction of movement is no longer a limitation. Experiment conducted using Bees simulation environment shows that the required bees have successfully moved to the identified location.**

Keywords: Bresenham, Object movement, Game theory and Bees Foraging.

I. INTRODUCTION

Bresenham Algorithm's [1] is a classical algorithm in computer graphics. Originally, it has been used to plot line between two pints. It approximates linear segments defined by rational coefficients using only integral points. There are many versions of Bresenham algorithm to speed up the algorithm performance and to optimize the time. In this paper, we proposed an algorithm which is inspired by Bresenham algorithm to be used in moving (in this case a Bee) an object between two locations. Moving object is different than plotting line in term of starting point and direction. In order to plot a line between two points, for example, "A" and "B", it does not matter from where to start because the target is plotting a line between the two points. Hence, the use of Bresenham algorithm for such a task would be sufficient. On the other hand, to move an object between to points requires more considerations - starting points and direction of movement. The adopted algorithm has to include different calculations for different directions and location. The proposed algorithm allows an object to be moved from any location to any direction with any slope degree. In other words, the new calculation can manipulate the location address of an object that is "X, Y" values in any orientation and move it to the new location address.

II. PREVIOUS WORK

There are many implementations of Bresenham's Algorithm in computer graphics. It has been implemented for drawing lines in raster device [2]. In their work, the line to be drawn always makes a non-negative angle not exceeding 45 degrees with the x-axis This results in a drastic reduction of computational time, nevertheless it limited to be used in a specific direction.

Bresenham's algorithm has also been used in Line Rasterization [3]. Zhang and Yang proposed a faster algorithm based on Bresenham's middle algorithm without line length limits. The paper emphasized on drawing lines on screen display with precision as the most important factor.

In [4], Bresenham's algorithm is modified to work on a half-plane and to be implemented with an 8-bit gate array slice. They present efficient algorithm to start with would be the one developed by Bresenham, which works in the first octant of the plane. In [5] a 3-D extension of Bresenham's algorithm and its implementation in the linear trajectory interpolation of CNC paths has been presented, the new approach will only work for lines with orientations between 0° and 45°. The new algorithm has been used to implement a CNC control system for a desktop milling machine and the results were acceptable for many applications requiring average precision and repeatability. The drawbacks are the low speed and power attainable due to the characteristics of those types of applications.

Mitchell A. Harris and Edward M. Reingold in their work [1] have applied Bresenham's algorithm with leap year calculations in different domain and approach, they compare the two calculations, explicate the pattern, and discuss the connection of the leap year/line pattern with integer division and Euclid's algorithm for computing the greatest common divisor, they show the relation between Bresenham's algorithm with leap year calculation.

Bresenham Line Algorithm has been used to draw 3D image in [6]. They have design an image chips for a high performance 3D rendering. The work is to accelerate 3D rendering algorithms based on Bresenham's line drawing and Pineda's parallel polygon drawing algorithms.

In [7], Jiang represents a vectorization method which is based on the Bresenham Line drawing algorithm. In this paper, the researcher represents a raster image by minimum number of line as an NP-complete problem. Their algorithm can serve as a special purpose compression algorithm which generates a portable vector file that can be uncompressed to the original raster image on any device that use the Bresenham line drawing algorithm.

Rokne and Brian Wyvill [8] have presented three improvement to the scan-conversion of lines. Their algorithm speeds up the scan-conversion (in the sense of the computational

effort) of line by a factor of roughly three over the original Bresenham version when the lines are sufficiently long.

Bresenham line algorithm has also been used in [9]. In their approach, the proposed approach outperformance the existing work by approximately 50% (the numbers present on the horizontal line between two of the data points show the percentage improvement). They have used Bresenham algorithm for computing lines for each pair of coordinates and compute FSM.

In [10], They present a 3D-Bresenham Rasterization in the framework of their VORTEX package and tested it on simulated structure grids. They showed in their preliminary study that Bresenham digitalization is a promising and worthwhile way to go.

III. Bresenham Algorithm

The Bresenham line algorithm is an algorithm which determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen.
[11].
Pseudocode:

```
function line(x0, x1, y0, y1)
int deltax := x1 - x0
int deltay := y1 - y0
real error := 0
real deltaerr := deltay / deltax
// Assume deltax != 0 (line is not
vertical),
// note that this division needs to
be done in a way that preserves the
fractional part
int y := y0
for x from x0 to x1
plot(x,y)
error := error + deltaerr
if abs(error) ≥ 0.5 then
y := y + 1
error := error - 1.0
```

It can be seen that Bresenham algorithm plots X and Y parameters only in one direction by increasing X in the for loop (`for x from x0 to x1`) and increase Y in `(y := y + 1)`. This algorithm is sufficient only for plotting a line because the requirement is straightforward. On the other hand, moving an object is different because there are occasion where the object needs to be moved in opposite direction; meaning the X and Y parameters should be decreased instead of increased. By considering other direction requirements (based on 8-point compass direction) we need six other different calculations. The proposed algorithm fulfils those requirements.

IV. The Proposed Algorithm

This algorithm uses four parameters (two parameters for the existing location (X1, Y1) of an object and the target location (X2, Y2) for the object). The direction of the movement will be determined by the location of the object from the target. The algorithm has eight parts (based on 8-point compass direction), each part caters for a specific direction in order to move an object from any position to the target.
Pseudocode:

```
Function move (x1, y1, x2, y2)
While(X1 != X2 Or Y1 != Y2)
   if (Y1 = Y2 And X1 < X2)
       X1 = X1 + 1; // Case 1
   if (X1 = X2 And Y1 < Y2)
       Y1 = Y1 + 1; // Case 2
   if (Y1 = Y2 And X1 > X2)
       X1 = X1 - 1; // Case 3
   if (X1 = X2 And Y1 > Y2)
       Y1 = Y1 - 1; // Case 4
   if (X1 < X2 And Y1 < Y2)
      double dx = Abs(X1 - X2);
      double dy = Abs(Y1 - Y2);
      rate += dy / dx;
      double iterate = dy / dx;
      if (rate > 1)
        for (int i = 0; i <= iterate; i++)
            Y1  = Y1 + 1;
            rate = 0;
      X1= X1 + 1;      // Case 5
  if (X1 > X2 And Y1 < Y2)
      double dx = Abs(X1 - X2);
      double dy = Abs(Y1 - Y2);
      rate += dy / dx;
      double iterate = dy / dx;
      if (rate > 1)
        for (int i = 0; i <= iterate; i++)
            Y1  = Y1 + 1;
            rate = 0;
      X1 = X1 - 1;     // Case 6
  if (X1 > X2 And Y1 > Y2)
      double dx = Abs(X1 - X2);
      double dy = Abs(Y1 - Y2);
      rate += dy / dx;
      double iterate = dy / dx;
      if (rate > 1)
        for (int i = 0; i <=    iterate;
        i++)
            Y1 = Y1 - 1;
            rate = 0;
      X1 = X1 - 1;     // Case 7
  if (X1 < X2 And Y1 > Y2)
      double dx = Abs(X1 - X2);
      double dy = Abs(Y1 - Y2);
      rate += dy / dx;
      double iterate = dy / dx;
      if (rate > 1)
        for (int i = 0; i <= iterate; i++)
```

```
            Y1 = Y1 - 1;
            rate = 0;
    X1 = X1 + 1;     // Case 8
```

A. *Experiment*

In order to realize the proposed algorithm, an experiment is undertaken in a simulation environment. The simulator is coded using Visual C# 2008. The objective is to model the foraging behaviour of *solitary bees*. In order to achieve such an objective, the bees need to collect food as efficiently as possible. We make the following basic assumptions:

- the environment consists of an infinite field of randomly distributed clumps of flowers of differing size and nectar yield (in the interests of simplicity, we will ignore the collection of pollen)
- the nectar yield of flowers varies throughout the day
- bees can see nearby clumps of flowers, but to determine the yield of the flowers the bee must land and sample the nectar
- bees can remember the location and yield of a previously visited clump of flowers and can accurately navigate to a remembered location
- the amount of nectar the bee can carry is limited
- bees fly at constant speed, only in daylight and can't fly all day
- flying requires more nectar than resting

Figure 1 shows an example of a Bee wanting to move to a flower. Such a desire will invoke case number 5 in the proposed algorithm because both of values X1 and Y1 are less than the targets values (X2, Y2).

Bee location = (X1 = 50, Y1 = 40);
Flower location = (X2 =90, Y2 = 160);
dx = Abs (X1 – X2) = | 50 – 90 | = 40
dy = Abs (Y1 – Y2) | 40 – 160 | = 120
slope = dy / dx = 120 / 40 = 3
slope = 3 ;

Based on the calculation, we obtain the slope value of 3. This means in order to move the object to the target, it has to increase the value of Y1 by 3 times and increase the value of X1 by only one time - *to move the Bee to the Flower, it needs 3 steps down (Y + 3) for each 1 step to right (X + 1).*

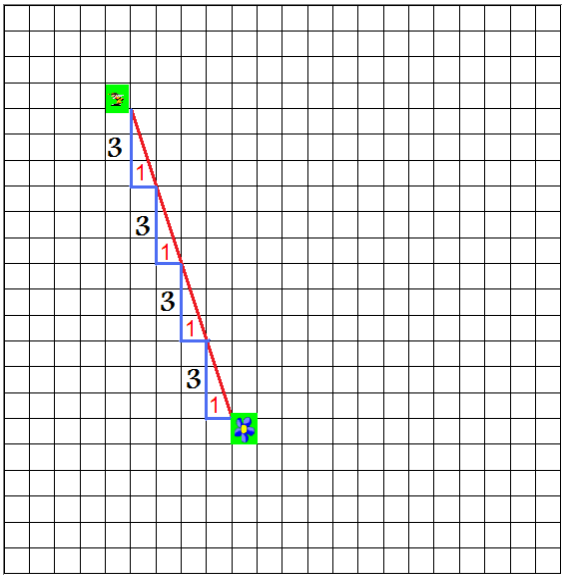

Figure 1: Example of Bee and Flower

Processing time is reduced when both objects (object to move and target) have the same value of (X1 = X2) or the same value of (Y1 = Y2). This is because such a requirement does not invoke a FOR LOOP while in other situations the algorithm uses FOR LOOP which takes extra time for calculation.

V. CONCLUSIONS

Existing work has shown that many types of application have using Bresenham algorithm, nevertheless it is done in different approaches. This paper demonstrates an algorithm based on Bresenham line drawing algorithm where the purpose is to move an object (for example a bee) to a targeted location (for example a flower or a hive). Such an approach have extended the use of existing Bresenham algorithm into a better use as direction of movement is no longer a limitation.

REFERENCES

[1] M. A. Harris, E.M. Reingold, "Line Drawing, Leap Years, and Euclid," ACM, 2004

[2] A. T. M. S. Khalid, M. Kaykobad, "An Efficient Line Algorithm," IEEE, 1997

[3] C. Z. Hua Zhang, Jun Yang, "Fast Algorithm for Line Rasterization by Using Slope 1," IEEE, 2005

[4] F. B. Pirri, O, "Interpolator engine for graphic applications," IEEE, 1987

[5] L. E. Chiang., "3-D CNC Trajectory Interpolation Using Bresenham's," IEEE, 1994

[6] M. W. Graham J. Dunnett, Paul E Lister, and Richard L. Grimsdale, "The Image Chip for High Performance 3D Rendering," IEEE, 1992.

[7] J.-H. C. Lijung Jiang, "Vectorization using Bresenham Lines," ACM, 1998.

[8] [8] B. W. J. G. ROKNE, "Fast Line Scan-Conversion," ACM, 1990.

[9] S. K. S. Hiren D. Patel, Reinaldo A. Bergamaschi, "Heterogeneous Behavioral Hierarchy for System Level Designs," ACM, 2006.

[10] M. s. S. a. Leonid I. Dimitrov, "Using 3D-Bresenham for Resampling Structured Grids," IEEE, 2004

[11] Wikipedia, "Bresenham's line algorithm," 2009.