An Approach of Line Scan Conversion Based on Multiple Segments

Aloke Kumar Saha and Kazi Shamsul Arefin

***Abstract*—The scan-converted straight line may contain many pixel segments of identical shapes. Therefore, instead of scan-converting the whole line step by step, we can scan-convert multiple segments of a line through copying and replicating. We have found that through shifting and copying we can significantly enhance the scan-conversion. In software simulation our result shows that on the average our new scan-conversion algorithm is always faster than existing algorithms.**

***Index Terms*—Pixel segment, GCD table, Floating point line, Slope table.**

I. Introduction

In computer graphics, scan-converting a straight line segment (or simply line) is the most basic operation. Many curves, wire frame objects, and complex scenes are composed of line segments. The speed of graphics rendering depends heavily on the speed of scan-converting a line. The ability to scan-convert a line quickly and efficiently is a very important factor in a graphical library. Bresenham's Midpoint algorithm (1965) [4,6] is the most important algorithm and has been widely used for line scan-conversion. In the past 45 years, many new methods have been proposed in the attempt to speed up the line scan-conversion process, and most of these new methods have been based on Bresenham's algorithm. Here we restrict our discussion to scan converting a straight line. Without creating any confusion, we inter-changeably use straight line, line segment, or simply line. We may divide existing line scan-conversion methods into six categories.

II. Scan-conversion Methods

A. Bresenham's Midpoint algorithm

Bresenham's algorithm [1] uses only integer operations to scan-convert a line on a plotter or any other equivalent graphics display device. The choice of pixels is made by testing the sign of a Discriminator based on the Midpoint principle. The Discriminator obeys a simple recurrence formula which can be calculated using only integer arithmetic and binary shift. When it begins to scan-convert the next pixel, it first modifies the Discriminator based on its original value by a few integer arithmetic and binary shift operations [7]. After that it tests the sign of this new Discriminator to decide which pixel should be selected. (The selected pixel is the closest to the actual line). The Discriminator sign testing approach is simple, robust and efficient. It can also be implemented in the hardware easily.

B. Our contributions

We observed that a scan-converted line may contain many identical pixel segments in their relative positions, as L0 shown in Fig. 1. We will show that if any two points on a line repeat their relative positions in the squares of the raster grid field then the line can be cut into segments and the corresponding pixel arrangements of the scan-converted line segments will repeat also. In other words, if (x0, y0) and (x0+r, y0+s) are on the line, where r and s are two arbitrary integers, then the corresponding scan-converted pixel arrangements from x= x0 to x= x0+r will be the same as the pixel arrangements from x= x0+r to x= x0+2r. Here (x0, y0) may or may not be integers. Therefore, multiple segments (or pixels) of the line can be replicated, or scan-converted in parallel.

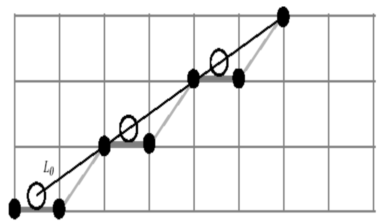


Fig. 1: Pixel segments having the same shape

As shown in Fig. 2, the length of the repeated segments (i.e., the repeating distance) on a line (L) can be decided by its translation (L1), of which one end point is on the raster grid (i.e., at integer coordinate). We will show that we only need additions and subtractions to find the distance between any pixel and the floating point line (L). Therefore in addition to replicating multiple segments of a line or scan-converting in parallel, we can achieve antialiasing easily and efficiently. In summary, we have major contributions in Multiple segments scan-conversion to speed up current line scan-conversion.

Manuscript received July 1, 2010.
Aloke Kumar Saha is with the Department of Computer Science and Engineering, University of Asia Pacific (www.uap-bd.edu), Dhanmondi, Dhaka-1209, Bangladesh. E-mail: aloke@uap-bd.edu.
Kazi Shamsul Arefin is with the Department of Computer Science and Engineering, University of Asia Pacific (www.uap-bd.edu), Dhanmondi, Dhaka-1209, Bangladesh. E-mail: arefin@uap-bd.edu.

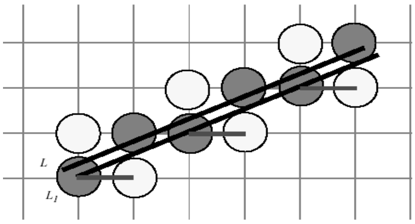


Fig. 2: Floating point line and antialiasing

Theoretically, we can speed up any existing line scan-converting algorithms up to m times, where m is the number of repeated line segments. In software simulation our result shows that on the average our new antialiasing algorithm is about 1.5 to 2 times faster than existing algorithms (Table 2). We have integrated Bresenham's Midpoint algorithm with our method to prove the idea. Hardware design diagram and some of the test statistics are provided.

C. Processes of Line Segmentation

LEMMA 1: Let $y = \frac{y_n}{x_n} = \frac{Pm}{Qm}$ x $= \frac{P}{Q}x$ be a line with integer end points (0, 0) and (xn, yn), where $0 \le x \le x_n$, $0 \le y \le y_n$, $y_n \le x_n$, $m \ge 1$, and GCD(P, Q)=1. (P, Q, and m are integers.) This line can be broken up into $m \ge 1$ segments, and each segment has Q pixels. The pixel arrangements of the m segments of the line take the same shape after scan-conversion. If dy=0, we define m=dx,Q=1, and P=0.

PROOF: From the equation of the line, we know that after x increases Q grids from x0=0, then y will increases P grids. So the point (Q, P) on this line is located on a grid. (0, 0) and (Q, P) are the two corresponding end points of the first and second segments. (See L0 in Fig. 1). The second segment starts from (Q, P), and the third segment starts from (2Q, 2P). Since the slopes of the line segments are the same, the pixel arrangements of the m segments are also the same. All the scan-converted segments of the line can be considered to be parallel translations of the first segment. The segments' end points are as follows:

segment 1: (0, 0) (Q-1, P-r);
segment 2: (Q, P) (2Q-1, 2P-r);
...;
segment m: ((m-1)Q, (m-1)P). (mQ-1, mP-r); and the ending point of the line: (mQ, mP)

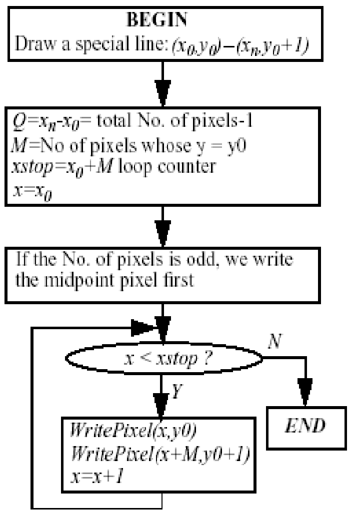
where r=round(P/Q). If we extend one extra pixel for each segment, we have:
segment 1: (0, 0) (Q, P);
segment 2: (Q, P) (2Q, 2P);
...;
segment m: ((m-1)Q, (m-1)P) (mQ, mP);
End of Proof.

Lemma 1 tells us that multiple segments (or pixels) of a line can be replicated, or scan-converted in parallel.

LEMMA 2: If P =1 and the line equation is: $y = \frac{y_n}{x_n}x = \frac{Pm}{Qm}x = \frac{1}{Q}x$, where $0 \le x \le x_n$ and $m \ge 1$, then each scan-converted segment (total m segments) has Q pixels, and there is only one grid unit in y direction between two neighboring segments. For each segment, the first $\lfloor Q/2 \rfloor + 1$ pixels have the same y value and the rest $\lceil Q/2 \rceil - 1$ pixels have also the same y value which is one grid unit more than the first $\lfloor Q/2 \rfloor + 1$ pixels' y value.

PROOF: We use the Midpoint principle in Bresenham's algorithm as the decision rule. Let's consider the first segment's pixels. For the first $\lfloor Q/2 \rfloor + 1$ pixels, the corresponding points (x, y) on the line have $x \le \lfloor Q/2 \rfloor$ and $y \le \frac{\lfloor Q/2 \rfloor}{Q}$. That is, $y \le 0.5$. According to the Midpoint principle, the first $\lfloor Q/2 \rfloor + 1$ pixels' y value will be 0. On the other hand, for the rest $\lceil Q/2 \rceil - 1$ pixels, x $\ge \lceil Q/2 \rceil$ and $y \ge \frac{\lceil Q/2 \rceil}{Q}$, thus we have $0.5 < y \le 1$.
Therefore, these pixels' y value will be 1.
End of Proof.

In this case, we don't need any compare operation. The algorithm to scan-convert a known line segment with this special condition is described as follows. Here the line doesn't necessarily start from (0,0).




```
DrawSpecialLine(int x0,int xn,int y0 int yn)
/*Scan convert a special line where yn=y0+1*/
{
  int I,xstop,Q;
  Q=xn-x0; /*total No of pixels - 1*/
  M=floor(Q/2)+1; /* No of pixels whose y=y0*/
  Xstop=x0+M;
  If(Q is even)
  {
    Xstop=xstop-1;
    Writepixel(xstop,y0);
  }
```

```
    for(i=x0;i<xstop;i++)
    {
      writepixel(I,y0);
      writepixel(i+m,yn);
    }
}
```

The discussions of the Symmetry principle in [2,3,4] are summarized as follows. We introduce the Symmetry principle here this is because we need to use it in our algorithms later.

LEMMA 3: Let f(x,y)=0 be a line with two end points (x0, y0) and (xn, yn). If any point (x', y') on the line has the same relative position in square of the grid field as that of the starting point (x0, y0), then if we translate the line such that (x0, y0) is on a grid point, (x', y') will also be on a grid point after the translation.

PROOF: Because the relative positions of the two points on the line in the grid field are the same, their relative positions in the grids will not change if they go through the same translation. As shown in Fig. 1, for example, P1 is the starting point of line L, P2 is a point on the line which has its relative position in the grids as P1. If we translate L to L0 such that P1 is at (0,0) on the grid, we can easily see that, after the translation, P2 is also on the grid.

End of Proof.

THEOREM 1: Let f(x,y)=0 be a line with two end points (x0, y0) and (xn, yn). If any point (x', y') on the line has the same relative position within a pixel (square) as that of the starting point (x0, y0), then the corresponding pixel arrangement of the scan-converted line starting from $\lfloor x' \rfloor$ will take the same shape as that starting from $\lfloor x_o \rfloor$. In other words, if (x0, y0) and (x0+r, y0+s) are on the line, where r and s are two arbitrary integers, then the shape of the corresponding scan converted pixel arrangements from x= $\lfloor x_o \rfloor$ to x= $\lfloor x_o \rfloor$ +r will repeat accordingly from x=$\lfloor x_o \rfloor$+r to $\lfloor Q/2 \rfloor + 1$ pixels' y value will be 0. On the other hand, for the rest $\lceil Q/2 \rceil - 1$ pixels, x $\geq \lceil Q/2 \rceil$and y $\geq \frac{\lceil Q/2 \rceil}{Q}$, thus we have 0.5<y ≤ 1. Therefore, these pixels' y value will be 1.

End of Proof.

In this case, we do not need any compare operation. The algorithm to scan-convert a known line segment with this special condition is described as follows. Here the line doesn't necessarily start from (0,0).

x=$\lfloor x_o \rfloor$+2r.

PROOF: Because the relative positions of point (x0, y0) and point (x', y') in the grids are the same, the Midpoint decision conditions are the same for the corresponding pixels, and therefore the pixel arrangement of the line from x= $\lfloor x_o \rfloor$ to x=$\lfloor x' \rfloor$ will also take the same shape as that from x= $\lfloor x' \rfloor$ to x= $\lfloor 2x' - x_o \rfloor$. The scan-converted pixel arrangement of the second segment will be exactly the same as that of the first segment if we treat (x', y') just as (x0, y0).

End of Proof.

According to Theorem 1, multiple segments of a line can be replicated, or scan-converted in parallel. Lemma 1 is a special case for the lines when their end points are all on the grids. Lemma 3 tells us how to find repeating segments of a line. Lemma 2 and Lemma 3 can help to speed up the scan-conversion of one segment. Next, we will discuss some modifications.

III. MODIFY EXISTING ALGORITHMS

A. Modify Bresenham's Algorithm

As mentioned in Lemma 1, if we can get m (the GCD of dx and dy), then we need only scan-convert the first dx/m pixels. The rest of the pixels can be scan-converted by copying and replicating the first dx/m pixels. However, if we calculate m, it requires a lot of time which may be more than the time spent by Bresenham's algorithm. Then we cannot really improve the algorithm. Here we modify Bresenham's algorithm, finding the GCD without calculating it directly. We use the terminology introduced in Foley, et. al's book [5]. Let y = Kx + C be a line with two integer end points (x0, y0) and (xn, yn), where K = $\frac{y_n - y_o}{x_n - x_o} = \frac{dy}{dx}$, $0 \leq k \leq 1$, $x_0 \leq x \leq x_n$, $y_0 \leq y \leq y_n$ and C is an integer. We can write the line in the implicit form f(x,y)=ax+by+c, where $x_0 \leq x \leq x_n$, a=dy, b=-dx and c=Cdx.

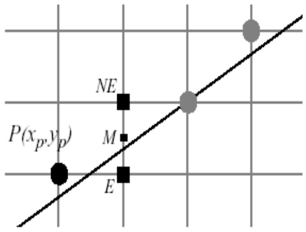


Fig. 3: Midpoint Discriminator: East or North-East

Suppose we have just selected point P(xp,yp) (Fig. 3), then the Discriminator of the Midpoint algorithm is: dold=f(xp+1,yp+1/2)=a(xp+1)+b(yp+1/2)+c.

If dold>0, then the NE grid is selected, and the next position we need to consider is (xp+2,yp+3/2). That is: dnew=f(xp+2,yp+3/2)=a(xp+1)+b(yp+1/2)+c+a+b=dold+a+b.

If dold ≤ 0, then the E grid is selected, and the next position we need to consider is (xp+2, yp+1/2), that is: dnew=f(xp+2,yp+1/2)=a(xp+1)+b(yp+1/2)+c+a=dold+a.

Because (x0,y0) is on the line, (i.e., f(x0,y0)=0), so we have: d0=f(x0+1,y0+1/2)=f(x0,y0)+a+b/2=a+b/2.

With a little modification to the Discriminator rules, we can decide whether the current point on the line is also on the grid. If the current point is on the grid, according to Lemma 1, we find a segment which can be used to replicate the other segments of the scan-converted line.

If dold>0, then the NE grid is selected. At the same time, if d'old=f(xp+1,yp+1)=dold+b/2=0, then we find the first segment's end point. This means (xp+1,yp+1) is on the line and is also located exactly on a grid. Therefore, we have the GCD: m=(xn-x0)/(xp+1).

If dold ≤ 0, then the E grid is selected. Similarly, if d'old=f(xp+1,yp)=dold-b/2=0, then we also find the first segment's end point. (xp+1,yp) is on the line and also on a grid. The GCD: m=(xn-x0)/(xp+1).

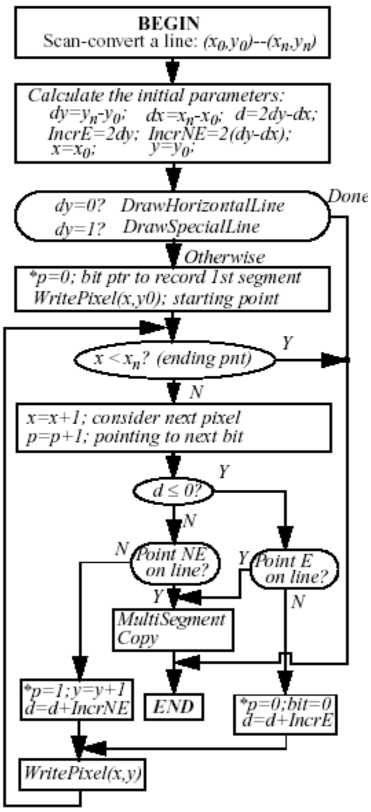
To avoid float point operation [1], we multiply f(x,y) by 2 and it does not affect the judgment. Now let us summarize the decision procedures:

d0=2a+b; ……………………….. (7)
If NE is chosen (i.e., dold>0), then
dnew=dold+2(a+b);……………..(8)
d'old = dold + b………………….(9)

If d'old=0, then (xp+1,yp+1) is on the line and also on a grid. We find the first segment's ending point.

If E is chosen (i.e., dold ≤ 0), then
dnew=dold+2a………………….(10)
d'old = dold – b………………...(11)

If d'old=0, then (xp+1,yp) is on the line and also on a grid. We find the first segment's ending point. Therefore with a little modification to Bresenham's algorithm, we can find the multiple segments of the line (Lemma 1). The algorithm is as follows:




```
Alg1_Software(int x0,int y0,int xn,int yn)
{ /* scan-convert a line: (x0,y0)--(xn,yn) */
  int dx,dy,incrE,incrNE,d,x,y; bit *p;

  x=x0; y=y0;
  dy=yn-y0; dx=xn-x0;
  d=2*dy-dx;
  incrE=2*dy; incrNE=2*(dy-dx);

  if (dy==0) {
    DrawHorizontalLine(x0,y0,xn); exit; }
  if (dy==1) {
    DrawSpecialLine(x0,y0,xn); exit; }

  p=seg; /* bit pointer to record the 1st seg */
  writepixel(x,y); /* starting point */
  while (x<xn) {
    x++; /* consider next pixel */
    if (d<=0) {
       if ((d-dx)==0) {MultiSegmentCopy; exit;}
       *p=0; d+=incrE; }
    else {
       if ((d+dx)==0) {MultiSegmentCopy; exit;}
       *p=1; y++; d+=incrNE; };
    writepixel(x,y);p++; /* write framebuffer */
  }
}
```

Here the bit pointer p is used to record the first segment's pixel positions, so that we can use it to make copies for the other m-1 segments. A bit = 0 means E is chosen; 1 means NE is chosen. It can be implemented in hardware (Section 4.) From the algorithm, if a line has n+1 pixels. We need compare operations. In Bresenham's Midpoint algorithm, there are n compare operations. If m=1 then we need n more compare operations than the Midpoint algorithm. But if m>2, then we need fewer comparisons than the Midpoint algorithm does. So for some lines, this algorithm may be slower. For many other lines, this algorithm is faster. In addition, with some extra cost, we can avoid many comparisons, as presented in the next section. In section 4, we will show that the multiple segments of the line can also be duplicated very quickly in the hardware.

IV. Hardware Design of Line Segment Replication

From the above discussion, we know that we can use segment copy operation to draw the other segments without any calculation. The speed of segment copy operation should be very fast. For one pixel wide lines, the hardware implementation will be expensive, because the addresses can be generated fairly fast by the original Midpoint algorithm, and the major limitation on speed is the memory bandwidth. However, for antialiased lines, it takes more time to calculate the intensities of several pixels in a column. The segment copy will significantly speed up the performance. The segment copy operation can be implemented in hardware easily without much cost. Fig. 4 is the hardware implementation diagram.

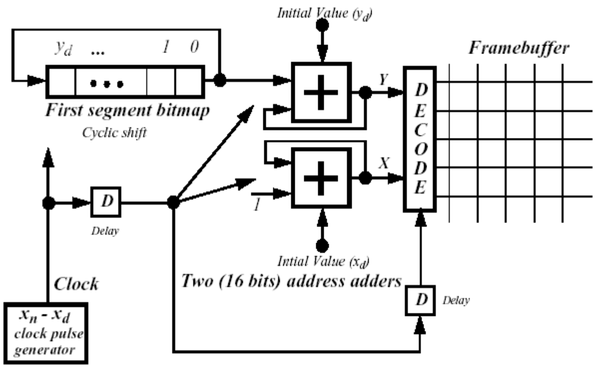


Fig. 4: Segment copy operation hardware implementation

In above diagram, (xd , yd) is the ending point of the first segment. Starting from the next pixel, x will add one at each clock pulse; y will depend on the bit pointer's cyclic shift bit value (0 or 1), corresponding to E or NE. The clock is initialized to generate xn -xd pulses, which corresponding to the number of the pixels to be replicated. The generated y address can be modified to y+i and y-i for many pixels in a column. The data bus will send the intensity of the corresponding pixels into the frame buffer. This work is done by hardware and only needs CPU to execute one or two I/O commands to start the hardware. The time spent by this copy operation is limited not by the copy operation hardware, but by the memory bandwidth. This portion is a conceptual design, which can be part of the future graphics acceleration hardware, separated from other parts. This design is not implemented.

V. Experimental Results

In the following table we will present the lists of all the pixels needed to be calculated and the speedup we can achieve for some lines (assuming the starting point of each line is at (0,0)).

TABLE 1: Sample lines considered and corresponding speedup

End-Point	No. of pixels scan-converted	Pixels Mid. Alg. calculated	Pixels Alg-I calculated	*Mid/A_I speedup*
(50,040)	801	500	25	*20*
-707,707	708	707	0*	-
-766,642	767	766	383	*2*
-866,499	867	866	866	*1*
-984,172	985	984	246	*4*
(99,687)	997	996	332	*3*
(100,010)	1001	1000	0*	-
(10,000)	1001	1000	0*	-
(1,023,197)	*1024*	*1023*	*1023*	*1*

Let's assume that all lines in a frame buffer area will appear to be scan-converted with the same probability. In the area enclosed by y=0, y=x, and x=N, we can draw at most (N+1)(N+2)/2-1 lines with different slopes and lengths (0 line_slope 1). The following table (Table 2) lists the total number of pixels which are considered by Bresenham's Midpoint algorithm and our new modified algorithm (in different square raster fields.) The last column is the total number of pixels the Midpoint algorithm considered divided by the total number of pixels our new algorithm considered, thus the speedup on the average we can achieve by drawing all these lines once. In practice, line segments need to be scan-converted inside a window, which on the average is shorter than the whole scan-converting grid area. From the experimental result of the Table 2, we can claim that the overall speedup is about 1.3-2 times faster than existing algorithm.

TABLE 2: Statistics of pixels visited by existing and modified algorithms

Area N+1	Total No. of Lines	Total No. of Pixels	Pixels Mid. Algorithm calculated	Pixels Algorithm-I Calculated	Mid/ A_I Speed up
2	2	4	2	-	-
5	14	54	40	7	5.71
8	35	203	168	66	2.55
10	54	384	330	142	2.32
20	209	2869	2660	1568	1.69
30	464	9454	8990	5673	1.57
56	1595	60115	58520	39221	1.49
100	5049	338349	333300	231539	1.44
150	11324	1136274	1124950	798855	1.41
200	20099	2686699	2666600	1908614	1.39
300	45149	9045049	8999900	6482436	1.39
400	80199	21413399	21333200	15407608	1.38
500	151524	55609674	55458150	40185222	1.38
670	224784	100478894	100254110	72734248	1.38
775	300699	155461899	155161200	112696242	1.38
810	328454	177475184	177146730	128747581	1.38
999	499499	332833499	332334000	241800057	1.37
1024	524799	358438399	357913600	260344616	1.37

"-" means that we need not any compare operation (according *Lemma 2.*)

VI. Conclusion

We have introduced a new method for scan-converting straight lines. Instead of scan-converting the whole line step by step, we can scan-convert multiple segments of a line through copying and replicating. We believe that our work is a significant contribution to implementing basic graphics primitives. We have plan to further investigate on this idea, and extend the method to curved lines and animation.

Acknowledgment

We would like to thank to Research and Publication Club (RPC) of Computer Science and Engineering (CSE) Department, University of Asia Pacific (UAP). This is because RPC always insists to all students and faculty members of UAP to publish their research papers. It bears all

expenditures of research activities and provides all sorts of support.

REFERENCES

[1] Bresenham, J. E., "Algorithm for Computer Control of Digital Plotter." IBM Syst. J., 4 (1965), 25-30.
[2] Bresenham, J. E., Grice, D. G., and Pi, S. C., "Run Length Slice Algorithms for Incremental Lines." IBM Technical Disclosure Bulletin, 22-813, 3744-3747, 1 (1980).
[3] Bresenham, J. E., "Run Length Slice Algorithms for Incremental Lines." in Fundamental Algorithms for Computer Graphics, Earnshaw R. A., Ed., NATO ASI Series, Springer Verlag, New York, 1985, 59-104.
[4] Gardner, P. L., "Modifications of Bresenham's Algorithm for Display." IBM Tech. Disclosure Bull. 18 (1975), 1595-1596.
[5] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F., Computer Graphics: Principles and Practice, Second Edition in C, Addison Wesley, 1996.
[6] http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
[7] http://www.cs.wustl.edu/~jain/cse567-08/ftp/scan.pdf
[8] http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

Aloke Kumar Saha is Head of Computer Science and Engineering Department of University of Asia Pacific (UAP), Dhaka, Bangladesh. He usually teaches courses on Digital Logic Design, Numerical Methods, Data Structures, Discrete Mathematics, Computer Graphics and Basic Electrical Engineering. His current research interests are Algorithm, Artificial Intelligence and Software Development. For more than 13 (Thirteen) years, he is also working with the undergraduate students of UAP, as a part of their thesis works, on the software development and implementation, Bi-Directional Heuristic Search Algorithm etc.

Kazi Shamsul Arefin has been serving as a Lecturer and Convener of Research and Publication Unit (RPU) with the Department of Computer Science and Engineering (CSE), University of Asia Pacific (UAP). He joined at UAP in October 2009 after completion his M.Sc. Engg. (CSE). Right now he teaches courses on Database Management System (DBMS), Software Development, and Programming on Artificial Intelligence and Expert Systems. Besides this, he is an Editor of International Journal of Computer and Information Technology (IJCIT), ISSN 2078-5828 (Print) and 2218-5224 (Online), www.ijcit.org. In addition to these, he is engaged in research activities throughout his undergraduate years and has 16 (sixteen) research papers (in 3rd IEEE ICCSIT, ICCIT, ICECC, ICEESD, IJECS, IJCTE, IJCEE, IJCIT and so on). Moreover, he is a member of International Association of Computer Science and Information Technology (IACSIT), Membership No: 80337708.