Depth Buffer DDA Based on FPGA

Fakhrulddin Hamid Ali
Computer Engineering Departement
University Of Mosul
Email: fhali310@yahoo.com

Abstract

The Digital Differential Analyzer (DDA) is normally used to efficiently compute the pixels (picture elements) for a straight line segment which can be used to represent it in a frame buffer or image memory. The calculated integer values of x and y for each pixel are used to address the memory while the color or intensity of the line segment presents the data to memory. The pixels in the frame buffer can then be read in a synchronized manner, while scanning the screen, and displayed on the computer monitor to show the straight line. This paper presents a new Digital Differential Analyzer as a three dimension (3D) version of the traditional (2D) one. There is a need to the 3D-DDA for the solution of the hidden surface problem in the image space when using depth or Z buffer method in the field of 3D computer graphics. A hardware implementation of the 3D-DDA is accomplished for the real time applications.

Keywords: DDA, pixel, depth buffer, scan-conversion.

المحلل التفاضلي الرقمي لذاكرة العمق باعتماد البوابات القابلة للبرمجة حقليا

فخرالدين حامد علي

قسم هندسة الحاسبات في جامعة الموصل – العراق

ألخلاصة

المحلل الرقمي التفاضلي يستخدم عادة لحساب النقاط الصورية لقطعة خط مستقيم وخزنها في ذاكرة الصورة حيث تستخدم القيمة المطلقة المحسوبة للاحداثي السيني و القيمة المطلقة المحسوبة للاحداثي الصادي لعنونة الذاكرة بينما يستخدم لون المستقيم كبيانات لها. هذه النقاط الصورية تقرأ من الذاكرة باسلوب متزامن اثناء عملية مسح الشاشة وتعرض عليها لتبيان صورة المستقيم. يقدم هذا البحث محللا تفاضليا رقميا جديدا ثلاثي الابعاد كتطوير للمحلل الرقمي التفاضلي التقليدي ثنائي الابعاد لجعل أدائه ثلاثي الأبعاد للاستفادة منه في حل مشكلة الأوجه المخفية باستخدام أسلوب ذاكرة العمق وذلك في تطبيقات توليد الصور للأجسام الثلاثية الأبعاد كما تم تنفيذ ذلك مباشرة بالكيان المادي لأغراض تحقيق الأداء في الزمن الحقيقي.

Received: 20 – 4- 2010 Accepted: 10 – 12 - 2010

1- Introduction

Pictures can be described in several ways. In computer graphic a raster display system is used to create and show these pictures, where a picture can be completely specified set of intensities for the pixel positions in display. A major task of the display process is digitizing a picture given in an application program into a set of pixel intensity values for storage in the frame buffer. This digitizing process is called scan-conversion. This requires access to the frame buffer for a huge number of times that makes it essential to adopt a fast scan-conversion algorithm in any real time system. However, the scan conversion of a polygon (a graphic major building primitive) is performed a line (or scan line) by line. The rasterization of a straight line segment can be accomplished using the line drawing algorithm called a Digital Differential Analyzer (DDA). On the other hand, it can also be done using Bresnham's algorithm (a modified DDA) which uses integer mathematics only. However, each of them operates in the image or 2D space and produces the same pixels for a line segment. Thus the segment is defined graphically by a set of points or pixels which lie between two end points or vertices. Since the application of depth or Z-buffer requires the value of the depth so it is required to generate this value for each pixel produced by the DDA. This motivates the conduction of this research work.

In 1987 Roman P.Molla designed three systems for different algorithms to implement scan conversion of a straight line segment. The Digital Differential Analyzer (DDA) and Bresenham algorithms were implemented using serial processing in addition to the implementation of the DDA algorithm using parallel processing. In his paper he discussed the performance , cost and the error ratio for the three designed systems mentioned previously [1]. In 1990 Arie Kaufman presented a paper introducing three dimensional scan-conversion algorithm, that scan-convert 3D parametric objects into their discrete voxel (volume element) map representation within a Cubic Frame Buffer (CFB). The parametric objects that are studied include Bezier form of cubic parametric curves, bicubic parametric surface patches, and tricubic parametric volumes. The converted objects in discrete 3D space maintain pre-defined application-dependent connectivity and fidelity requirements. The algorithm introduced imply third-order forward difference techniques. Efficient versions of the algorithm based on first-order decision mechanisms, which employed only integer arithmetic, are also discussed. All algorithms are incremental and use only simple operations inside the inner algorithm loops. They perform scan-conversion with computational complexity which is linear in the number of voxels written to the (CFB) [2] . In 1993 the researchers Andreas Schilling and Wolfgan Straber introduced an algorithm that deals with hidden surface elimination problem at pixels level. The algorithm provided the solution of the aliasing problem resulted in the scan conversion operation. The hardware implementation was divided into three stages in order to apply the pipeline technique to improve the performance. The architecture designed from 12000 gates and the chip has ability to produce 20 M pixel/sec [3]. In 1997 Marc Olano and Trey Greer presented a new triangle scan conversion algorithm that works entirely in homogeneous coordinates. By using homogeneous coordinates, the algorithm avoids costly clipping tests which make pipelining or hardware implementations of scan conversion algorithms difficult. The algorithm handles clipping by the addition of clip edges, without the need to actually split the clipped triangle. Furthermore, the algorithm can render true homogeneous triangles, including external triangles that should pass through infinity with two visible sections [4]. In 2001 Hans Holten-Lund described in his thesis useful methods and techniques for designing scalable hybrid parallel rendering architectures for 3D computer graphics. Various techniques for utilizing parallelism in a pipelined system are analyzed and a prototype 3D graphics architecture named Hybris has been developed. Working hardware / software code design implementations of Hybris for standard-cell based

ASIC (simulated) and FPGA technologies have been demonstrated. Using manual co-synthesis for translation of a virtual prototyping architecture specification written in C into both optimized C source for software and into to a synthesizable VHDL specification for hardware are implemented. A 3D medical visualization workstation prototype (3D-Med) is examined as a case study and an application of the Hybris graphics architecture [5]. In 2003 the researchers Khun Yee Fung, Tina M. Nicholl, and A. K. Dewdney presented a new run-length slice algorithm although run-length slice algorithms are seldom used because of the division operation required. The biggest advantage of the basic algorithm is the reduction of additions used which is considered outweighed by the division required. In this paper, the new run-length slice algorithm that does not require a division operation is presented. Furthermore, it uses the double-stepping paradigm in incremental line drawing algorithm to reduce the number of additions used by at least half [6]. In 2004 David Harris discussed the performance of one of the OpenGL library units. This unit is the lighting unit which is responsible for light simulation and the brightness. The lighting unit requires the floating point mathematics which needs a very high processing time. He has introduced a hardware implementation for this unit using integer mathematic operations only, where the designed architecture consisted of multipliers and look up tables [7]. In 2005 the researchers Sven Woop, Jörg Schmittler, and Philipp Slusallek presented a paper describing the architecture and a prototype implementation of a single chip, fully programmable Ray Processing Unit (RPU). It combines the flexibility of general purpose CPUs with the efficiency of current GPUs for data parallel computations. This design allows for real time ray tracing of dynamic scenes with programmable material, geometry, and illumination shaders. Although, running at only 66 MHz, the prototype FPGA implementation already renders images at up to 20 frames per second, which in many cases beats the performance of highly optimized software running on multi-GHz desktop CPUs. The performance and efficiency of the proposed architecture is analyzed using a variety of benchmark scenes [8]. In 2006 D. Wang et al presented an antialiasing method using a DSP-based display system for removing the undesired jaggies occurred in the line drawing. It is concluded in this paper that the application of antialiasing on color lines takes 60 times the time required without antialiasing [9].

Beside this section, section 2 presents the hidden surfaces concept supported by the depth buffer algorithm presented in figure(1). Sections 3 and 4 contain the algorithm of 3D-DDA and its implementation using FPGA while the test results are in section 5. Section 6 contains conclusions and the references are listed at the end of the paper.

2- Hidden Surfaces Concept

A major consideration in the generation of realistic images for 3D objects is the identification and removal of the parts of a scene that are not visible from a chosen viewing position. Faces are classified as back faces, which are entirely invisible, and front faces which can be completely visible when they are un covered (not obscured). On the other hand, front faces can be covered (obscured), partially or entirely, by some part of a 3D object.

The test for and elimination of a back face is direct using the normal to the plane of the face. The detection and elimination of covered front faces, or parts of them, is more complicated. Many approaches can be taken to solve this problem and numerous algorithms have been devised for the identification of invisible objects, or parts of them, for different types of applications. Some methods require more memory , some involve more processing time , and some apply only to special types of objects. The various algorithms are referred to as visible surface detection or hidden surface elimination methods[10]. They are broadly

classified according to whether they deal with object definition directly or with their projected images or with both [11].

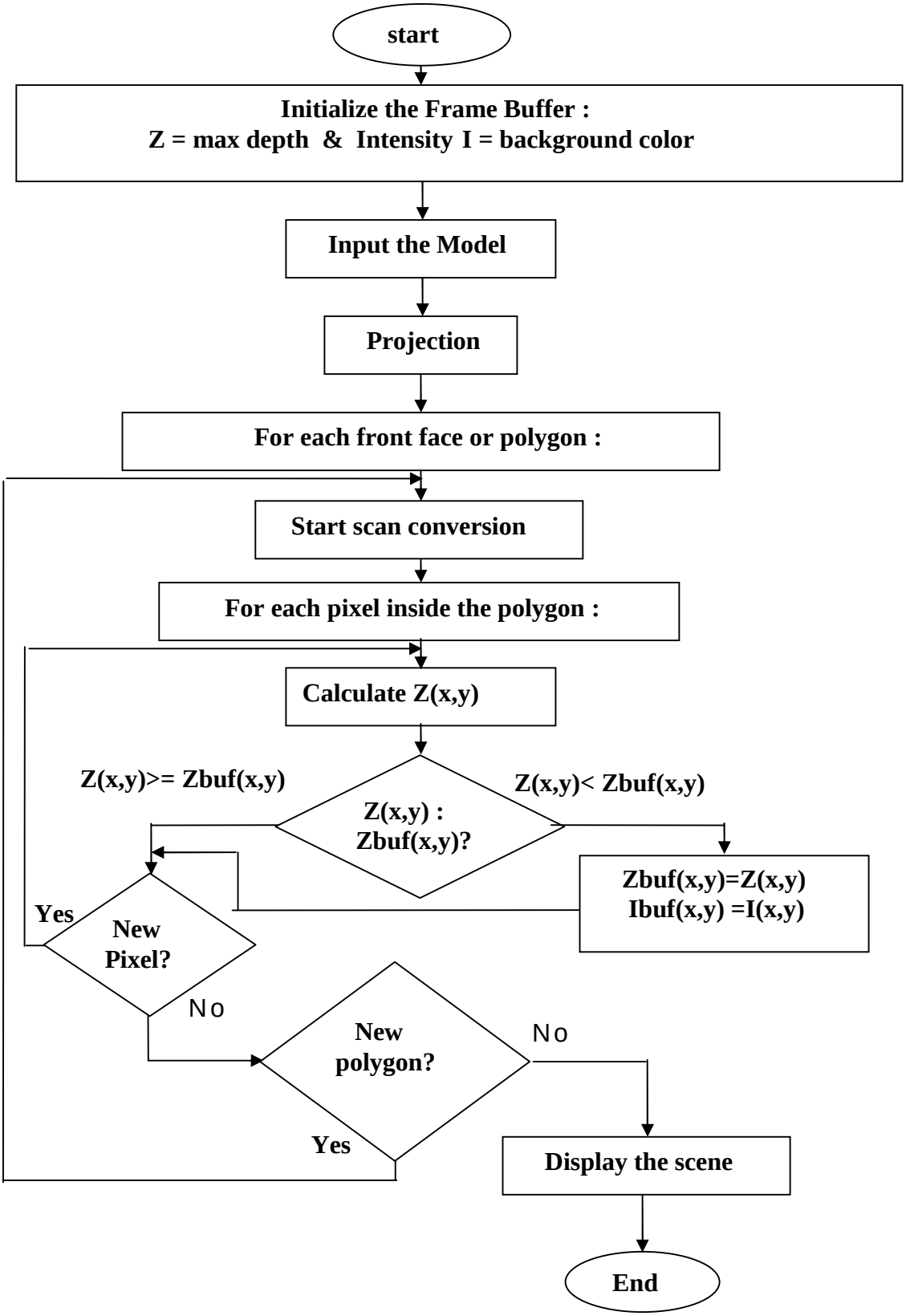


Figure (1): Depth buffer algorithm

A commonly used image space method for detecting invisible surfaces is the depth buffer method [12]. This method (also called z buffer) involves solving the hidden surface problem by storing the depth of the nearest pixel location to the viewer. A depth buffer (pair of intensity and z values) can be used to display images and remove hidden surfaces by simply scan converting all front faces, in a random order, and updating a pixel value only when the depth of a point that projects on the pixel is less than the depth stored in the buffer. The buffer requires to be initialized to the background intensity or color and maximum Z before updating a new image. Aversion of this method is illustrated in figure (1). The major disadvantage of this method is that it requires additional memory to store the depth or Z value for each pixel. Semiconductor memories have become cheaper and denser which encourage adopting this method. OpenGL emerged from silicon graphic which is a software interface to a graphic hardware and is a low-level graphics library specification which is becoming a famous standard in the computer graphics field has also adopted this method [13].

Due to the above advances, the DDA requires to be developed and its operation extended to work in the 3D space. The implementation of the three dimensional digital differential analyzer (3D-DDA) in hardware is also essential for real time applications.

3- Three Dimensional DDA (3D-DDA)

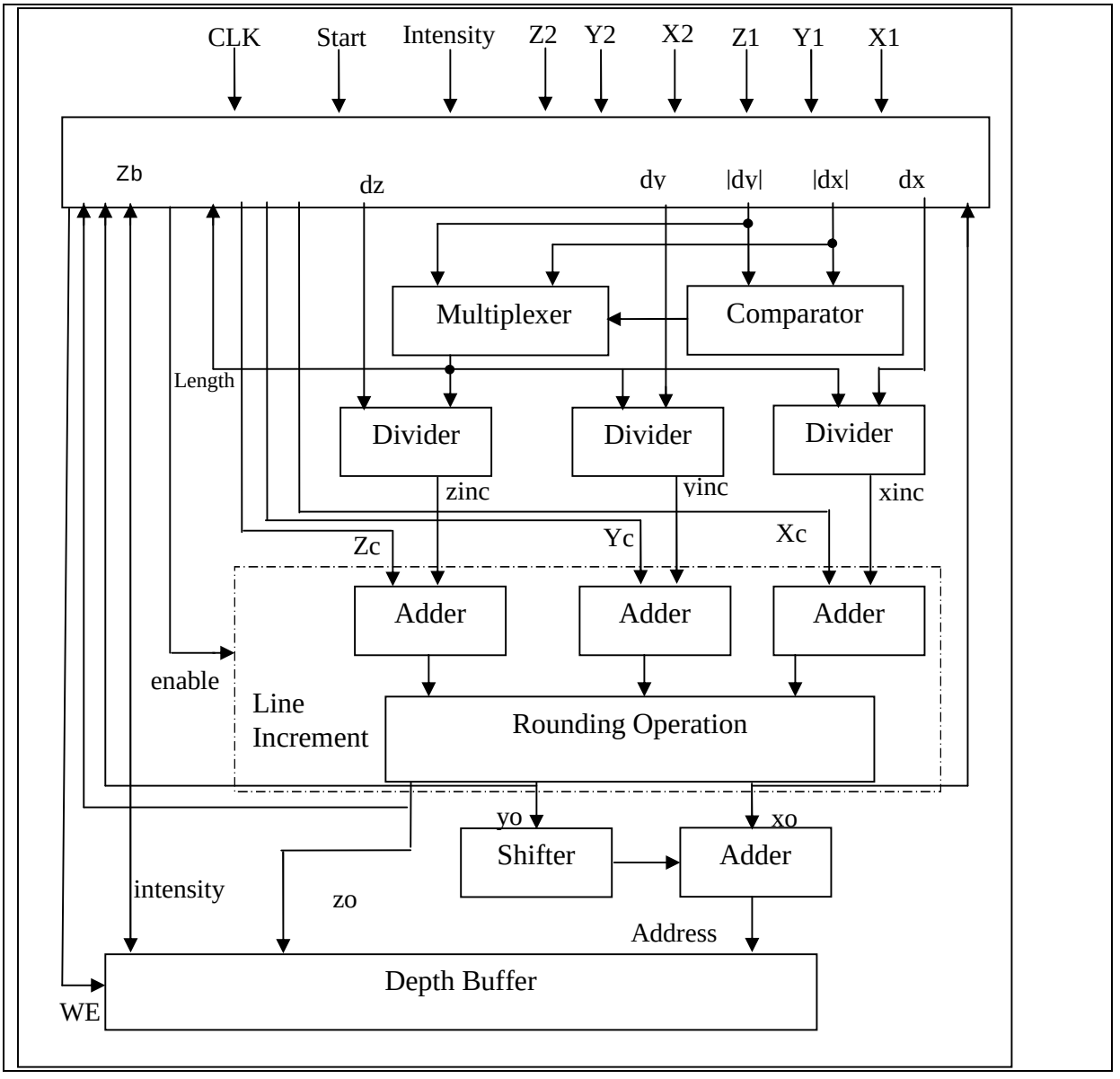
A new three dimension version of Digital Differential Analyzer is accomplished by considering the line segment whose pixels require to be generated in three dimensional space. So for each pixel a z value is calculated in addition to the x and y values so that the algorithm works in the object space rather than in the image space. The proposed algorithm for the 3D-DDA becomes like :

```
Enter the line segment end points (x1,y1,z1) and (x2,y2,z2)
Length = abs(x2 – x1)
  if  abs(y2 – y1)  > Length then
      Length = abs(y2 – y1)
increment (x) = (x2 – x1) / Length
increment (y) = (y2 – y1) / Length
increment (z) = (z2 – z1) / Length
Begin Loop
 i = 1
 while (i ≤ Length)
    store pixel(Round(x),Round(y),Round(z))
    x = x + increment (x)
    y = y + increment (y)
    z = z + increment (z)
    i = i +1
 end while
End Loop
```

4- FPGA implementation of 3D-DDA

In this section a hardware implementation, using FPGA, of the 3D-DDA is presented. A block diagram of the hardware designed unit is illustrated in figure (2). The scan

conversion operation begins when the start signal is set "ON". The control unit part is responsible for calculation of the difference in x, y, and z coordinates, in addition to that it enables the line increment part. The scan conversion operation of a line segment requires its two input vertices V1(X1,Y1,Z1) and V2(X2,Y2,Z2) with its intensities as inputs to the hardware unit. The computed greatest coordinates absolute difference (dx or dy) is then, using division operation, used to calculate the increment value of x, y, and z. After that the intermediate pixels are calculated using addition operation, each time the increment value is added to x , y, and z coordinates. Then the rounding operation is performed to produce integer values. The address of the frame buffer is calculated for each pixel from its computed coordinate values (Xc,Yc) to load the intensity data and z value at the right location in the buffer.



Figure(2): 3D-DDA implemented unit

5- Test results

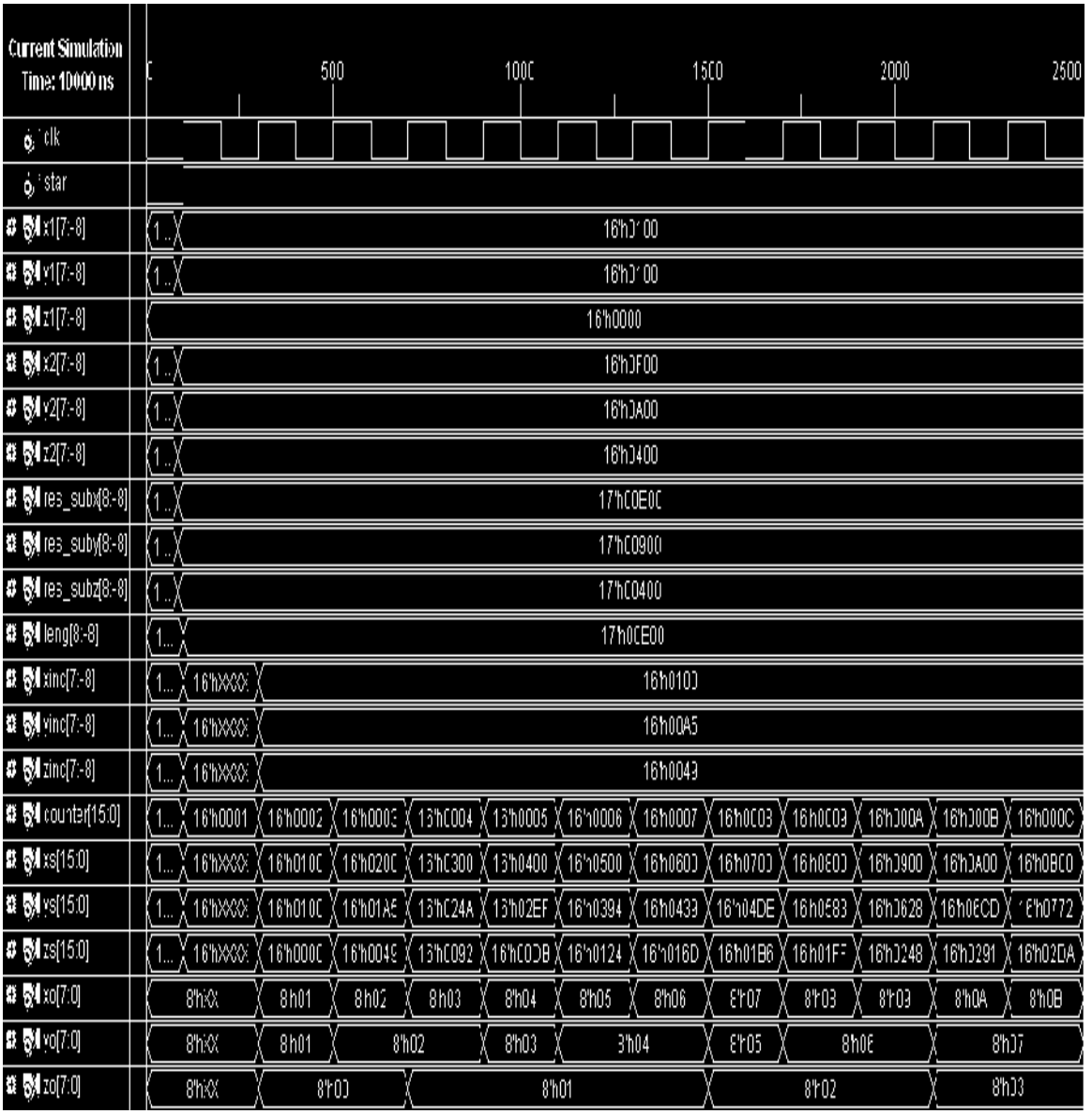
The 3D-DDA is implemented using VHDL and synthesized using FPGA available on the kit-board Spartan-3E. Two testing examples are used for verification. Figure (3) shows the simulation waveforms of example (1) executed by the implemented hardware where:

star: start signal.

x1, y1,z1 & x2, y2,z2: the two end vertices of the line segment.
res_subx , res_suby & res_subz : the coordinate differences dx, dy, and dz respectively, where: dx = (x2-x1), dy = (y2-y1), and dz = (z2-z1).
leng: the greatest absolute coordinates difference dx or dy.
xinc: increment in x coordinate.
yinc: increment in y coordinate.
zinc: increment in z coordinate.
xs: the current x coordinate before rounding.
ys: the current y coordinate before rounding.
zs: the current z coordinate before rounding.
xo: the current x coordinate after rounding.
yo: the current y coordinate after rounding.
zo: the current z coordinate after rounding.

In figure(3) the inputs vertices for test example (1) are (01.00 h ,01.00 h ,00.00 h) and (0f.00 h ,0a.00 h, 04.00 h) fixed point representation (8 bit for the integer and 8 bit for the fraction). The scan conversion operation begins when the 'star' signal becomes "ON". The control unit computes the coordinate difference values dx (0e.00), dy (09.00), and dz (04.00). Then the greatest coordinate difference (absolute dx or dy) is determined to evaluate the length of the line (leng =0e.00 h). After that, the increment in x ,y and z are calculated (xinc=01.00 h which is equal to 1, yinc=00.a5 h which is equal to 0.644, and zinc=00.49 h which is equal to 0.2851) by dividing dx, dy, and dz by the length. From these values the increment in x, the increment in y, and the increment in z coordinates are used to determine all pixels between the two vertices and at each pixel (xs, ys, zs) rounding operation is performed. However, the simulation of the first step values in figure (3) are calculated theoretically according to the 3D-DDA algorithm, for comparison, in the following steps:

dx = x2-x1 = 0f.00 – 01.00 = 0e.00 hex.
dy = y2-y1 = 0a.00 – 01.00 = 09.00 hex.
dz = z2-z1 = 04.00 – 00.00 = 04.00 hex.
Leng =0e.00 hex.
Xinc = dx/leng = 01.00 hex = 1.
Yinc = dy/leng = 00.a5 hex =0.644.
zinc = dz/leng = 00.49 hex = 0.2851.
xs = x1+ xinc = 01.00 + 01.00 = 02.00 hex.
Xo = rounding(02.00) = 2.
ys = y1+ yinc = 01.00 + 00.a5 = 01.a5 hex.
Yo = rounding(01.a5hex) = rounding(1.644) = 2.
zs = z1+ zinc= 00.00 + 00.49 = 00.49 hex.
Zo = rounding(00.49 hex) = rounding(0.2851) = 0.



Figure(3): Simulation sample results of example 1

The incremental calculation of pixels continues till the last one which is the second end or vertex of the straight line segment. To verify the performance of the designed unit, the pixels are theoretically computed and listed in Table (1).

Table(1) Theoretically computed pixels for example 1

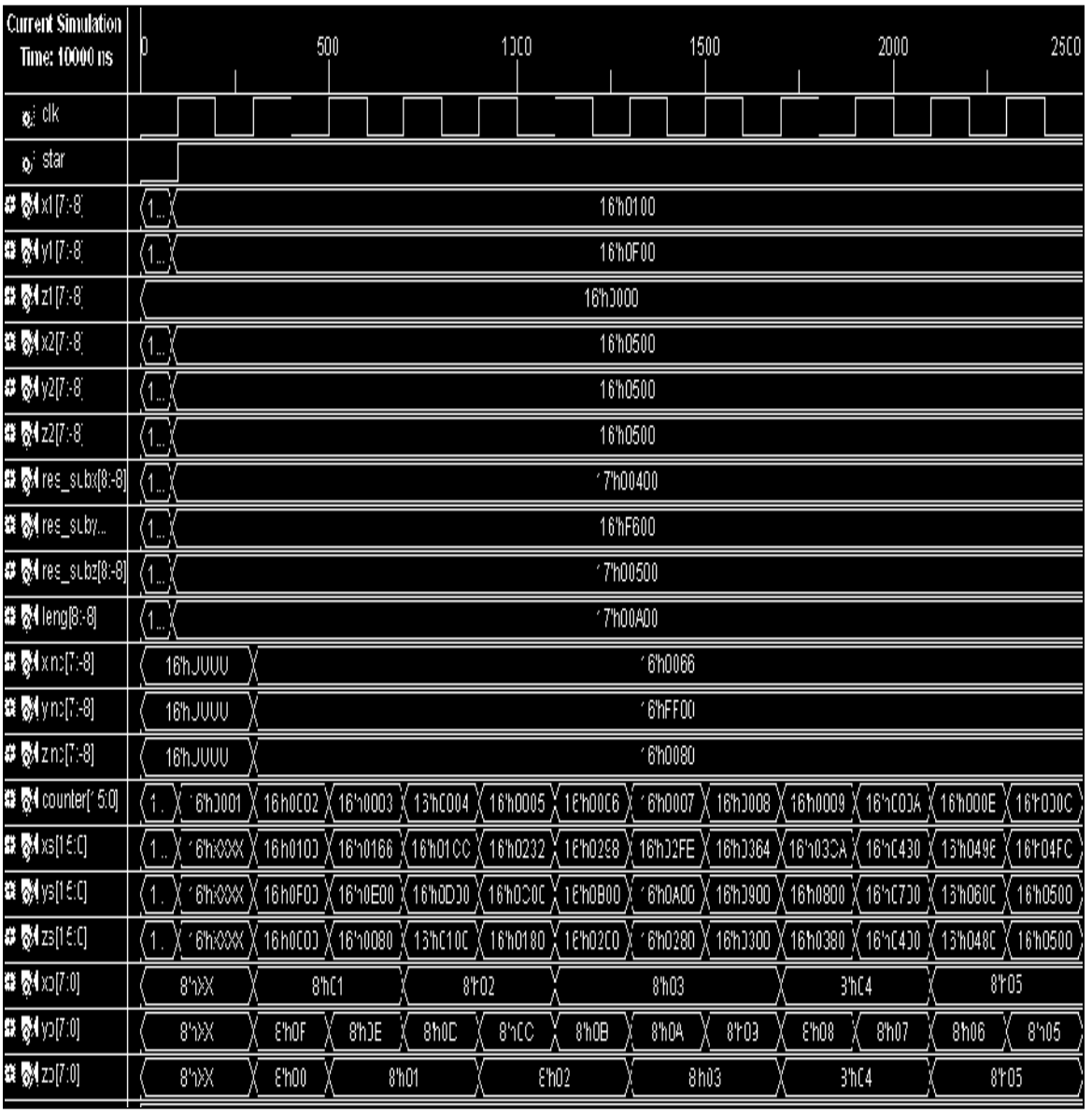
x	y	Rounded y	z	Rounded z
1	1	1	0	0
2	1.6428	2	0.2857	0
3	2.2857	2	0.5714	1
4	2.9285	3	0.8571	1
5	3.5714	4	1.1428	1
6	4.2142	4	1.4285	1
7	4.8571	5	1.7142	2
8	5.5000	6	2.0000	2
9	6.1428	6	2.2857	2
10	6.7857	7	2.5714	3
11	7.4285	7	2.8571	3
12	8.0714	8	3.1428	3
13	8.7142	9	3.4285	3
14	9.3571	9	3.7142	4
15	10.0000	10	4.0000	4

In figure(4) the inputs vertices for test example (2) are (01.00 h ,0f.00 h ,00.00 h) and (05.00 h ,05.00 h, 05.00 h). The coordinate difference values dx is equal to 04.00h, dy is equal to f6.00h (equivalent to -10), and dz is equal to 05.00h. So the length of the line is equal to 0a.00h (10 is the absolute value of dy). After that, the increment in x ,y and z are calculated (xinc = 00.66 h which is equal to 0.398437 (approximately 0.4), yinc is ff.00 h which is equal to -1, and zinc=00.80 h which is equal to 0.5) by dividing dx, dy, and dz by the length. Other calculations are:

xs = x1+ xinc = 01.00 + 0.66 hex = 01.66 hex. = 1.398
Xo = rounding(1.398) = 1.0
ys = y1+ yinc = 0f.00 - 01.00 = 0E.00 hex. = 14.0
Yo = rounding(0E.00hex) = rounding(14) = 14.0
zs = z1+ zinc= 00.00 + 00.80 hex = 00.80 hex. = 0.5
Zo = rounding(00.80 hex) = rounding(0.50) = 1.0

Next values are
Xo = 01.66 hex + 0.66 hex = 02.32 hex. = 2.195
Xo = rounding(2.195) = 2.0
Yo = E hex -1.0 = D hex = 13.0
zs = 00.80 + 00.80 hex = 1.60 hex. = 1.375
Zo = rounding(1.60 hex) = rounding(1.375) = 1.0

Other values are incrementally computed in the same manner.



Figure(4): Simulation sample results of example 2

6- Conclusions

A new algorithm as a three dimensional development of the available two dimensional Digital Differential Analyzer is designed and implemented using the configurable Field Programmable Gate Array (FPGA). This is accomplished by computing the depth value or z and storing it in the frame or depth buffer, in addition to the intensity, for each pixel so that the application of depth buffer method for removing the hidden surfaces becomes feasible. The calculation if z is done incrementally rather than using the 3D line equation for each pixel. By this way, the hardware is simpler and operates faster. The designed unit can accept straight line segments with negative slope which produces negative dx or negative dy as demonstrated by example 2. So the unit is capable of working with different slope angles and locations. The representation of pixel values using two hexadecimal digits (or 8 bit) before the decimal and two hexadecimal digits after (or 8 bit) has no effect on the accuracy. This is

because each pixel coordinate fraction value is rounded to the nearest integer value before being stored in the frame buffer as illustrated by table (1). Table (2) shows the utilization resources of Spartan3 Kit that is used to implement the unit. The hardware unit can produces pixels at a speed of 120 M pixel per second assuming a very small time is lost in computing the increment values (one cycle as shown in the waveforms), before the production of pixels, which slightly reduces the maximum operating frequency in the table.

Table(2) Resources utilization for the 3D-DDA unit for 128*128 pixel (2 bytes per pixel, one for intensity and one for Z)

Type Resources (or Frequency)	Utilized Resources	Total Resources	Ratio
Number of Slices	662	4656	14%
Number of Slices Flip Flops	1049	9312	11%
Number of 4 input LUTs	852	9312	9%
Number of Bounded IOBs	73	232	31%
Number of Block RAMS	16	20	80%
Number of MULT18X18s	0	20	0%
Number of GCLKs	1	24	4%
Maximum Operating Frequency	120.389 MHZ		

References

[1]: Roman P.Molla Vaya , “Parallel Fixed Point Digital Differential Analyzer “ , Computer Journal , Vol. 23, No. 1, pp: 46-52, 1987.

[2]: Arie Kaufman ,” Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes” , International Conference on Computer Graphics and Interactive Techniques , ,ISBN:0-89791-227-6, pp: 171 – 179, 1990.

[3]: Andreas Schilling , Wolfgang Straßer , “EXACT: Algorithm and Hardware Architecture for an Improved A-Buffer”, Computer Graphics, vol. 27, no. 4, (SIGGRAPH ’93 Proceedings), pp: 85–92, August 1993.

[4]: Marc Olano, Trey Greer ,“Triangle Scan Conversion using 2D Homogeneous Coordinates”, SIGGRAPH / EUROGRAPHICS Conference On Graphics Hardware Proceedings of the ACM SIGGRAPH / EUROGRAPHICS workshop on Graphics hardware, ISBN:0-89791-961-0, Pages: 89 – 95, 1997.

[5]: Hans Holten-Lund, “Design for Scalability in 3D Computer Graphics Architectures”, Ph.D. thesis Computer Science and Technology Informatics and Mathematical Modelling Technical University of Denmark, July, 2001.

[6]: Khun Yee Fung, Tina M. Nicholl A. K. Dewdney,” A Run-Length Slice Line Drawing Algorithm without Division Operations”, Computer Graphics Forum Volume 11 Issue 3, pp: 267 – 277, Feb 2003.

[7]: David Harris, “An Exponentiation Unit for an OpenGL Lighting Engine”, Member, IEEE , IEEE TRANSACTIONS ON COMPUTERS, VOL. 53, NO. 3, pp: 251-256, March 2004.

[8]: Sven Woop, Jörg Schmittler, and Philipp Slusallek,” RPU: A Programmable Ray Processing Unit for Real Time Ray Tracing”, SIGGRAPH In Proceedings of the ACM SIGGRAPH /EUROGRAPHICS Conf. on Graphics Hardware, pp:27–36, 2005.

[9]: Dusheng Wang , Hock Lye Toh , Xiang Chen , Fan Yang ," A Simple Anti-Aliased Method For Straight Line Drawing Based On DSP Platform ", Posters proceedings ISBN 80-86943-04-6 , WSCG' 2006 , January 30 – February 3, 2006.

[10]- David Blythe, Scott R. Nelson," Lighting and Shading Techniques for Interactive Applications ", SIGGRAPH 99, Course 12, Art. 5,pp: 37, August 8, 1999.

[11]- Sherif Ghalib Max Planck, Institute for Computer Science, Saarbrucken, Germany, "Object Space Visibility", SIGGRAPH, course # 6, 2006.

[12]- James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, and Richard L. Phillips, " Introduction to Computer Graphics ", Addison _ Wesley, 13th printing, pp 451, July 2000.

[13]- Mark Segal , Kurt Akeley, " The OpenGL Graphics System : specification ", Silicon Graphics Inc., 2002.

The work was carried out at the college of Engineering. University of Mosul