Hardware Implementation of 3D-Bresenham's Algorithm Using FPGA

Dr. Basma Mohammed Kamal Younis, Lecturer Ne'am Salim Mohammed Sheet
Department of Computer Technology Engineering-Technical College-Mosul

Abstract

Traditional 3D-Bresenham's algorithm is efficient in generating lines on raster systems using only integer calculations. This algorithm is needed as a solution of hidden surface problem using depth-buffer method to calculate ***z*** value for each pixel, while calculated values of ***x*** and ***y*** are used to address frame buffer memory, ***z*** value is used to test hidden surface by saving the closest depth in depth buffer.

In this paper Bresenham's algorithm for plotting 3D-lines is examined then modified to simplify hardware requirements during implementation phase. Basing on efficiency of the algorithm on the space symmetry an enhanced version of this algorithm is implemented using OpenGL. Experimental results confirm results calculated theoretically for both traditional and modified algorithms.

The hardware implementation is accomplished for real time applications, and a graphic sub-system is designed using FPGA. Finally, a comparison is accomplished for Spartan3E utilization which is used to implement the hardware unit.

Keywords: Computer graphics, Bresenham, Pixel, Scan conversion, FPGA.

تنفيذ الكيان المادي لخوارزمية برزنهام ثلاثية الأبعاد باستخدام مصفوفة البوابات المبرمجة حقلياً

الخلاصة

تعد خوارزمية برزنهام التقليدية خوارزمية كفوءة تستخدم في توليد الخط المستقيم ثلاثي الأبعاد في الأنظمة النقطية باستخدام حسابات القيم الصحيحة فقط. كما أن هذه الخوارزمية تقوم بحل مشكلة الأوجه المخفية التي تستخدم طريقة ذاكرة العمق لحساب قيمة البعد الثالث لكل نقطة شاشة. تستخدم قيم الاحداثي السيني والصادي لحساب عناوين ذاكرة الصورة وقيمة البعد الثالث لفحص الأوجه المخفية بواسطة حفظ العمق الأقرب للناظر في ذاكرة العمق.

في هذا البحث تم دراسة خوارزمية برزنهام لرسم الخطوط ثلاثية الأبعاد وتطويرها لتناسب متطلبات مرحلة التنفيذ بالكيان المادي. اعتماداً على خاصية التناظر تم تحسين الخوارزمية باستخدام مكتبة الرسم المفتوحة المعروفة بـ(OpenGL) حيث تم مطابقة النتائج العملية والنظرية لكلتا الخوارزميتين التقليدية والمطورة.

تم تنفيذ الكيان المادي لخوارزمية برزنهام ثلاثية الأبعاد بعد تطويرها لأغراض تحقيق الأداء في الزمن الحقيقي، حيث تم تنفيذ منظومة الرسم الفرعية باستخدام البوابات القابلة للبرمجة حقلياً. وأخيراً تم إجراء مقارنة بين المعماريتين المصممتين من ناحية الاستفادة من مصادر رقاقة Spartan3E المستخدمة للتنفيذ المادي.

الكلمات الدالة:رسومات الحاسوب،برزنهام، نقطة شاشة، تحويل المسح، مصفوفة البوابات المبرمجة حقلياً.

Introduction

Computer graphics remains one of the most existing and rapidly growing computer fields. Computer graphics may be defined as a pictorial representation or graphical representation of objects in a computer. In computer graphic a raster display system is used to create and

show the pictures of objects, where in raster display system The rasterization is process of determining the appropriate pixels for representing picture or graphics object, a picture can be completely specified set of intensities for the pixel positions in display, Picture definition is stored in a memory area called frame buffer or refresh buffer [1].

Line drawing is one of the most fundamental activities in computer graphics. There are many different line drawing algorithms used in computer graphics. Application of inefficient algorithms may cause drawing to require unacceptably large amount of time thus making the graphics presentation boring. This is why algorithms used in computer graphics must be computationally very efficient. As a result computer graphics algorithms are quite often found to avoid, for example, floating point operations or more costly division and multiplication operations as in Bresenham's line algorithm. As a line is drawn by lighting only a finite number of pixels with integer coordinates, it is not possible to produce a theoretical line exactly in a raster device. In order that a line drawn on a raster device simulates a theoretical line as closely as possible, a set of pixels, which represent the real line as closely as possible, are only switched on[2].

However, the scan conversion of a polygon (a graphic major building primitive) is performed by a scan line method. The rasterization of a straight line segment can be accomplished using any line drawing algorithm. In this work the value of depth is determined for each pixel produced by the 3D Bresenham algorithm, for depth or z-buffer application. Following review of some related published works:

In 1991 Edward Angle and Don Morrison present that a Bresenham's algorithm is the standard for scan converting a line segment. A version based on the properties of linear Diophantine equations can speed up scan conversion by a factor of almost five [3].

Also A. T. M. ShafiquI Khalid and M. KaykobadZ in 1996 present a new algorithm for drawing lines in a raster device in which a suitable data structure has been chosen to avoid comparisons that are required, for example, in Bresenham's algorithm. Experimental results as well as clock cycles calculated theoretically suggest that this new algorithm outperforms the ones currently existing in the literature in terms of computational time. Their experimental results also suggest that quality of the line does not deteriorate even when high resolution raster devices are used [2].

A group of researchers in 2004 designed a system effectively implemented two different algorithms for calculating the intermediate points in a line given the two endpoints, and representing the fundamental elements of this system in VHDL and using the available FPGAs, their first algorithm is the Digital Differential Analyzer (DDA) which requires floating-point intermediate values and the second is the Midpoint Line Algorithm, a special case of Bresenham Line Algorithm, which is famous for its speed and accuracy [4].

Andre Redert propose in 2004 a depth scaling method that enables visualization of arbitrary-shaped 3D scenes on 3D displays , his approach uses spatially adaptive depth scaling that maximizes the perceptual 3D effect, from the original scene geometry , the topology and local depth ordering among objects are preserved , while depth linearity is discarded [5].

In 2006 S. Fawad reviewed the basis of Bresenham algorithm in graphic interpolation processes. There are doubtless other areas where straightforward interpolations across

polynomials can be managed using this technique. It seems to be a reasonable approach to teaching interpolation processes, even though there may be faster algorithms for many of these interpolations [6].

In 2009 Niu Lianqiang and Feng HaiWen presented A new fast line drawing algorithm that is different from the traditional Bresenham algorithm, A line is treated as an aggregation of several line segments and the y coordinate differences of candidate pixel points in every step of traditional algorithm are replaced by the length errors of each segments in this new algorithm. Each operation and judgment can generate a line segment by keeping the advantages of integer arithmetic and then the numbers of operating and output are decreased. Besides these, the skew symmetric character is considered in the algorithm and the direct draw property without operation of some special lines is also pointed out [7].

Also in 2011 Chikit Au and Tony Woo uncover the reason for little prior works. The concept of the mid-point in a unit interval generalizes to that of nearest neighbors involving a Voronoi diagram. In their paper, the three-dimensional extension is based on the main idea of Bresenham Algorithm of minimum distance between the line and the grid points. The structure of the Voronoi diagram is presented for grid points to which the line may be approximated. The deployment of integer arithmetic and symmetry for the three-dimensional extension of the algorithm to raise the computation efficiency are also investigated [8].

In 2011 Fakhrulddin Hamid Ali designed a new algorithm as a three dimensional development of the available two dimensional Digital Differential Analyzer (DDA) and implemented it using the configurable Field Programmable Gate Array (FPGA), In his paper he concluded that the hardware unit can produces pixels at a speed of 120M pixel per second assuming a very small time is lost in computing the increment values [9].

Theory of 3D-Bresenham's Algorithm

The original approach of Bresenham's algorithm for plotting a two-dimensional line between origin point (x_a, y_a) to a end point (x_b, y_b) is adopted to present the three-dimensional ,for plotting a three-dimensional line between origin point (x_a, y_a, z_a) to a end point (x_b, y_b, z_b).

The traditional 2D-Bresenham line generation algorithm is shown in Figure1.

The three dimension version of Bresenham's Algorithm is accomplished by considering the line segment whose pixels require to be generated in three dimensional spaces. 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 3D-Bresenham algorithm and the modified one are shown in Figure2 and Figure3 respectively. As we can see from Figure3 that the loop calculate the line pixels is know one instead of three in the flowchart of Figure2 which will effect on the amount of hardware component in the implementation part.

These algorithms are tested using OpenGL for many possible line orientation and the generated pixels values are checked, the vision results are shown in Figure4 where they are the same for the two versions of algorithm.

Practical System Implementation

A block diagram of the designed hardware unit of the 3D-Bresenham is shown in Figure5, where the scan conversion operation of a line segment

requires it's two input vertices start vertex v_1 (x_a, y_a, z_a) and end vertex v_2 (x_b, y_b, z_b) as an input to the hardware unit, then compute the greatest coordinate difference (*dx, dy* and *dz*) with the error value to calculate the increment value of *x, y, z*. After that the intermediate pixels are calculated each time the increment value is added to the *x, y,* and *z* coordinate.

The address value of the frame buffer is calculated for each pixel from its computed coordinate(*x, y*) to load the intensity data (RGB) in the buffer.

The refresh controller access the frame buffer periodically to obtain the data necessary to refresh the monitor and display the image stored in the frame buffer [10]. The refresh controller unit shown in Figure8 generates the timing and synchronization signals. The *HS* and *VS* signals are control the horizontal and vertical scans of the monitor. It generates the video-on signal to indicate whether to enable or disable the display to display the form which is only within the dimensions of the screen.

The graphic controller accesses the frame buffer to update the image. The basic operation of the graphic controller in this work is one of scan line method, 3D-Bresenham's algorithm, to set the pixel intensity values for storage in the frame buffer.

The arithmetic section of the implemented graphic controller starts from entering the vertices of start and end point of the line and compute the slope of each *y, x* and *z*. This section computes the corresponding address value of the frame buffer using *x,* and *y* coordinates and also computes *z* value of each pixel in the line with its intensity, Figure6 and Figure7 show a block diagram of the designed graphic controller of the 3D-Bresenham's algorithm and the modified one respectively.

The calculated integer values of *x* and *y* for each pixel are used to address the memory (frame buffer) while the color(RGB) or intensity of the line segment presents the data to be storing in the frame buffer with its associated address. 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.

The tradeoff between the access of the refresh controller and the access of the graphic controller is a key idea for the architecture of many graphic systems. The current design, as shown in Figure5, overcomes this problem by using dual port frame buffer memory [10], the frame buffer here is performed using block RAM. Physically, the block RAM has two completely independent access ports, labeled Port A and Port B. The structure is fully symmetrical, and both ports are interchangeable and support data read and write operations. Each memory port is synchronous with its own clock, clock enable, and write enable. Read operations are also synchronous and require a clock edge and clock enable [11].

This work drains all the capacity of on-board block RAM (XC3S500E) of Spartan3E, as used 640 by 480 VGA screen. Where the address of each pixel is equal to: *y-pixel * 640* plus *x-pixel*. So maximum pixel the address is (*479*640 + 639*) which is equal to *300kb*, so were exhaust all the capacity of block RAMs in Spartan3E FPGA kit used in this work were it is *360kb*.

Test results and discussion

The 3D-Bresenham's algorithm is synthesized using VHDL and implemented using FPGA available on the kit-board Spartan-3E. Many testing examples are used for verification where are the same as OpenGL results, following example illustrates one of this

tests samples then Figure9 shows the simulation waveforms that obtained by the implemented hardware.

Example1:

To verify the performance of the designed unit, the pixels are theoretically computed and listed below the steps as mentioned in 3D-Bresnham's algorithm in Figure2:

Step1: enter the two end vertices of the line segment.

x_a,y_a,z_a=(25,10,4),x_b,y_b,z_b= (20,20,0).

x-addr =25 , *y-addr* =10 , *z-addr* =4 , as shown in the simulation in Figure9, which represent the initial value of pixel address, but in the algorithm in Figure2 it consider as (*x, y, z*).

Step2: the coordinate differences *dx, dy,* and *dz* respectively, where *dx=(xb-xa),dy=(yb-ya),*and*dz=(zb-za).*
Then dx = -5 , dy = 10 ,dz =-4 .

Step3: enter *dx, dy* and *dz* in a comparison with zero.
If (dx < 0), yes dx = -5, so *xinc* = -1.
If (dy < 0), no dy = 10, so *yinc* = 1.
If (dz < 0), yes dz = -4, so *zinc* = -1.

Step4: enter one of the three condition while dx=-5, dy=10, dz=-4.

Since |dy|>|dx| and |dy|>|dz|, so the middle condition is verified as mentioned in the algorithm in Figure 2.

Step5: compute *err1* and *err2*:
err1=2*|dx|-|dy|=2*|-5|-|10|=2*5-10=0.
err2=2*|dz|-|dy|=2*|-4|-|10|=2*4-10 = -2.

Step6: compare *err1* and *err2* with zero. If (*err1*>0),update *err1* and *x-addr*, but this condition is not true then pass it and check if(*err2*>0),update *err2* and *z-addr*, this condition is not true then pass it and jump to the next step.

Step7: update the following variable:
err1=err1+2*|*dx*|=0+(2*5)=10.
err2=err2+2*|*dz*|=-2+(2*4)=6.
y-addr=y-addr+yinc= 10+1=11.

Step8: use a temporary register *m* as shown in Figure9 that indicate the number of pixel, and update it incrementally.

Step9: check the condition if *m<dy* (maximum slope), then repeat the steps from 6 to 9 until *m=dy* (stop condition).

Example2:

To verify the performance of the designed unit, the pixels are theoretically computed and listed below the steps as mentioned in Modified 3D-Bresnham's algorithm in Figure3:

Step1: enter the two end vertices of the line segment.

x_a,y_a,z_a=(25,10,4), x_b,y_b,z_b= (20,20,0)

Step2: the coordinate differences *dx, dy,* and *dz* respectively, where
dx=|x_b-x_a|,*dy*=|y_b-y_a|,*dz*=|z_b-z_a|.
dx=|20-25|=5,*dy*=|20-10|=10,*dz*=|0-4|=4.

Step3: enter one of the three condition while *dx*=5, *dy*=10, *dz*=4.

Since *dy>dx* and *dy>dz*, so the middle condition is verified as mentioned in the algorithm in Figure3, so *flag*=1, and an exchange will happened (*dy* by *dx*, x_1 by y_1 , x_2 by y_2).

Step4: *x*=10, *y*=25, *z*=4, which represent the initial value of pixel address.

Step5: enter the new coordinate x_1, x_2, y_1, y_2, and z_1, z_2 in a comparison.
If (x_1>x_2), no x_1=10, x_2=20, so *xinc*=1.
If (y_1>y_2), yes y_1=25, y_2=20, so *yinc*=-1.
If (z_1>z_2), yes z_1= 4, z_2=0, so *zinc*=-1.

Step5: compute *err1* and *err2*:
err1=2**dy-dx*=2*5-10=2*5-10=0.
err2=2**dz-dx*=2*4-10=2*4-10=-2.

Step6: compare *err1* and *err2* with zero. If (*err1*>0),update *err1* and *y-addr* then check *err2*. If (*err2*>0), update *err2* and *z-addr* then jump to the next step. But the two conditions are false then jump to the next step.

Step7: update the following variable:
err1=err1+2**dy*=0+(2*5) =10.
err2= err2+2**dz*=-2+(2*4)=6.
x = x + xinc=10+1=11.

Step8: check the *flag*, as we determined in Step3 *flag*=1, so *x-addr=y, x-addr=x, z-addr=z*.

Step9: use a temporary register *m* as shown in Figure9 that indicate the number of pixel, and update it incrementally.

Step10: check the condition if *m<dx* (maximum slope), then repeat the steps from 6 to 9 until *m=dx* (stop condition).

Table1 illustrates that OpenGL results for this example is the same as the VHDL simulator results.

Also in this paper we have applied all the possibility of line segment in the hardware unit successfully, as shown in Figure10.

Table (2) and Table (3) shows the utilization resources of Spartan3E Kit that is used to implement the hardware unit. The difference in the two hardware units is that the Bresenham algorithm unit can produces pixels at a speed of 76M pixels per second but the Modified 3D-Bresenham's algorithm can produces pixels at a speed of 68 M pixels per second , assuming a 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 Table2 and Table 3.

Conclusions

From the above analysis it is clear that performance of the Bresenham's algorithm depends on the largest value from *dx*, *dy* or *dz*, in the modified algorithm we try to shrink the algorithm to make hardware simpler. The code for the modified algorithm is shorter in length than that the old 3D-Bresenham's algorithm. So hardware requirements for the modified one are also less. Hardware implementation for the modified is much cheaper than the other algorithm.

Our proposed technique can efficiently be used for the improvement of other computer graphics primitive's algorithm which use Bresenham's algorithm, and can be used in many applications such as calculating z-buffer values to improve the traditional hidden surface removal method as shown in the work.

The designed system effectively implemented using FPGA two different versions of scan line method (Bresenham's algorithm) that generalized to three-dimensional for using in scan conversion of a polygon in 3D-scene.

In designing this system, it was illustrated how the drawing of a simple 3D-line is more complex than initially thought. Simulation illustrated the issues that arise upon calculating a line and reinforced the fact that the three-dimensional algorithm used to raise the computation efficiency since it use integers only. At the end we improved a simple hardware and the designed unit can also accept all the type of slope (negative and positive) in efficient way.

References

1. Donald Hearn and M. Pauline Baker ,M.Pauline Baker, *"Computer Graphics, C version"* , 2nd edition. Prentice Hall, Inc. 1997.
2. T. M. ShafiquI Khalid and M. Kaykobad,*"an Efficient Line Algorithm"*, Journal of Circuits and Systems, IEEE 39th Midwest symposium, Vol. 3, Pages: 1280-1282 , 1996.
3. Edward Angle and Don Morrison ,*"Speeding Up Bresenham's Algorithim"*, university of new mexico, November 1991 , IEEE Computer Graphics & Application.
4. Jong Lorraine, Shirachi Lisa and Wang Sherman, *"Computer Graphics: Where Straight Lines, Aren't"*, Computer Science Department University of California, Final Project, Winter Quarter 2004.
5. Andre Redert ,*"Visualization of Arbitrary-shaped 3D Scenes on Depth-limited 3D Display"* , Journal

of 3D Data Processing, Visualization and Transmission, 2004 IEEE Proceedings. 2nd International Symposium, 938-942 , 2004.

6. S.Fawad, *"Adapting Bresenham Algorithm "*, Journal of Theoretical and Applied Information Technology ,Vol. 2 Issue: 2 , 27-30, 2006.
7. Niu Lianqiang and Feng HaiWen ,*"A Line Segments Approximation Algorithm of Grating Lines"*, Journal of 2009 International Forum on Computer Science-Technology and Applications, IEEE Computer Society , Vol. 2 , 34-37, 2009.
8. Chikit Au and Tony Woo, *"Three Dimensional Extension of Bresenham's Algorithm with Voronoi Diagram"*, Journal of Computer-Aided Design, Vol. 43, Issue: 4 , 417-426, 2011 .
9. Fakhrulddin Hamid Ali, *"Depth Buffer Depth Buffer DDA Based on FPGA",* Journal of Al-Rafidain Engineering ,Vol.19 , No.5 , October 2011.
10. Fakhrulddin Hamid Ali and Amar I. Dawod, *"FPGA Design and Implementation of a Scan Conversion Graphical Sub-System",* Journal of Al-Rafidain Engineering, Vol.16 ,No.4, Oct. 2008.
11. Xilinx Company, *"Spartan-3 Generation FPGA User Guide"*, June 25, 2008.
12. J. E. Bresenham, *"Algorithm for Computer Control of a Digital Plotter"*, IBM Systems Journal, Vol.4 , no. 1 , 25-30 1965 .
13. Edward Angel, *"Interactive Computer Graphic: A Top- Down Approach Using OpenGL"*,Addition Wesley, Third Edition 2003.
14. F.S. Hill , Jr ,*"Computer Graphics Using OpenGL"*, second edition, Prentice Hall International, 2001.
15. Xilinx Company, *"User Manual Spartan-3 FPGA Family:Complete data Sheet",* March 4, 2004.
16. Xilinx Company, *"Spartan-3E FPGA Starter Kit Board User Guide"* , June 20, 2008.
17. Xilinx Company,*"Spartan-3E FPGA Family: Data Sheet"*,August 26, 2009.

Tikrit Journal of Engineering Sciences/Vol.20/No.2/March 2013, (37-47)

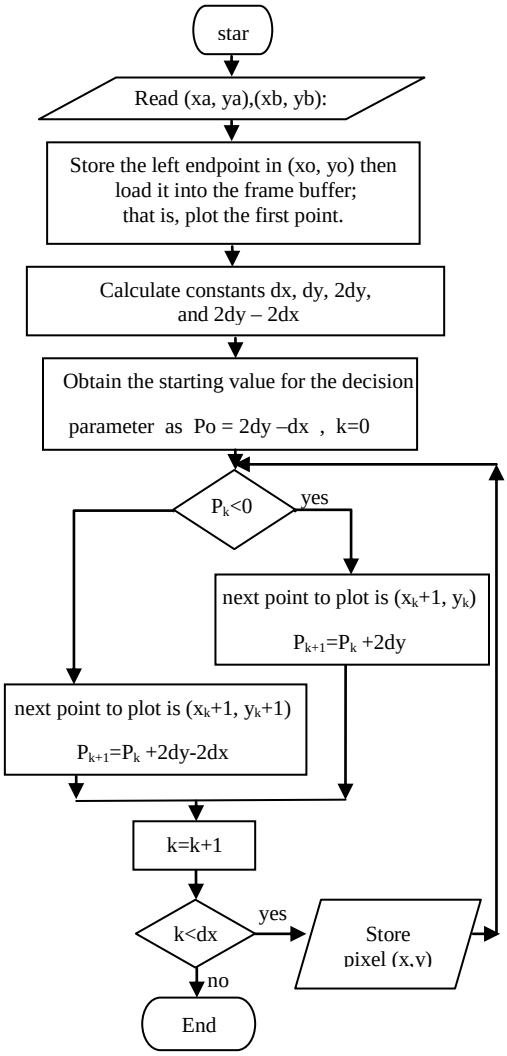

Figure 1: 2D-Bresenham algorithm

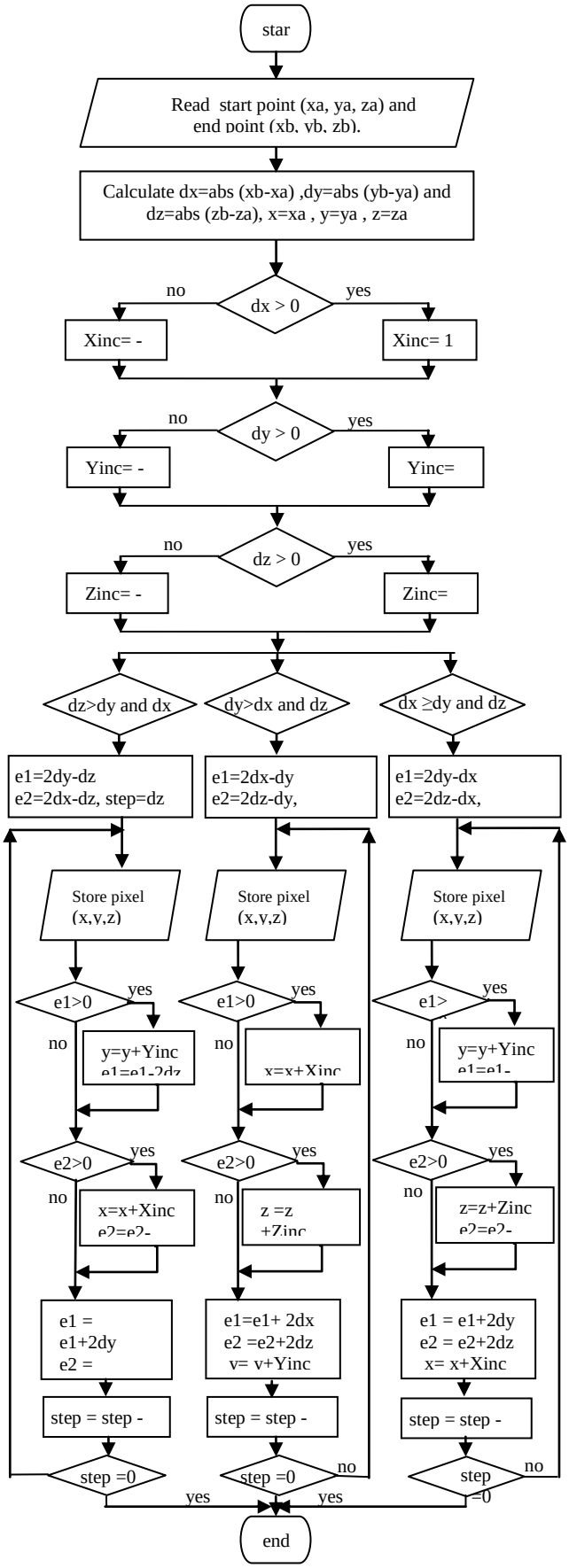

Figure 2: 3D_Bresenham algorithm

Tikrit Journal of Engineering Sciences/Vol.20/No.2/March 2013, (37-47)

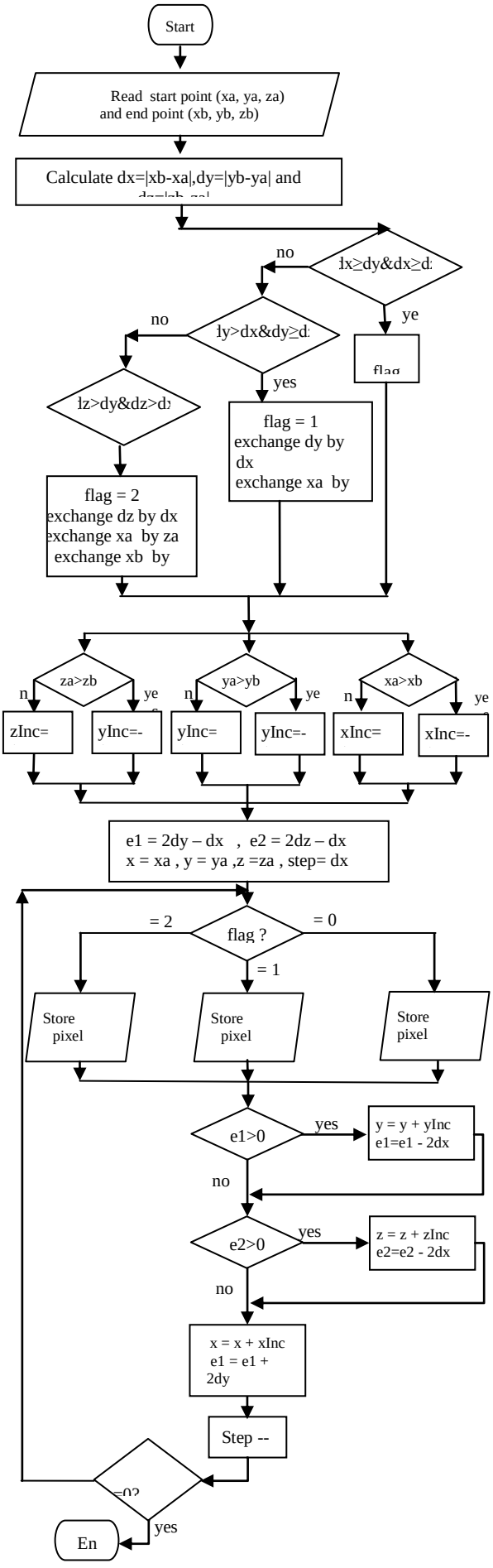

Figure 3: the Modified 3D_Bresenham algorithm

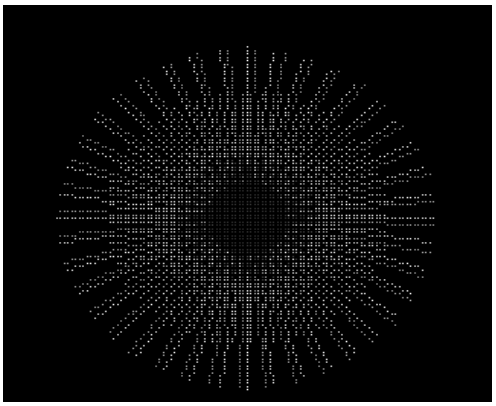

Figure 4: OpenGL results that Displays a Series of 3D_Lines Fanned in a spherical shape

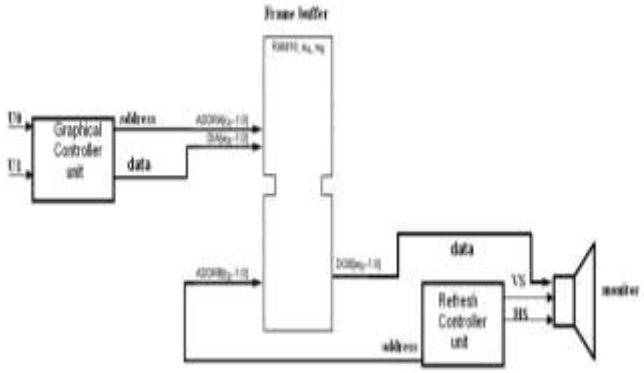

Figure 5: Hardware graphic sub-system in FPGA

Tikrit Journal of Engineering Sciences/Vol.20/No.2/March 2013, (37-47)

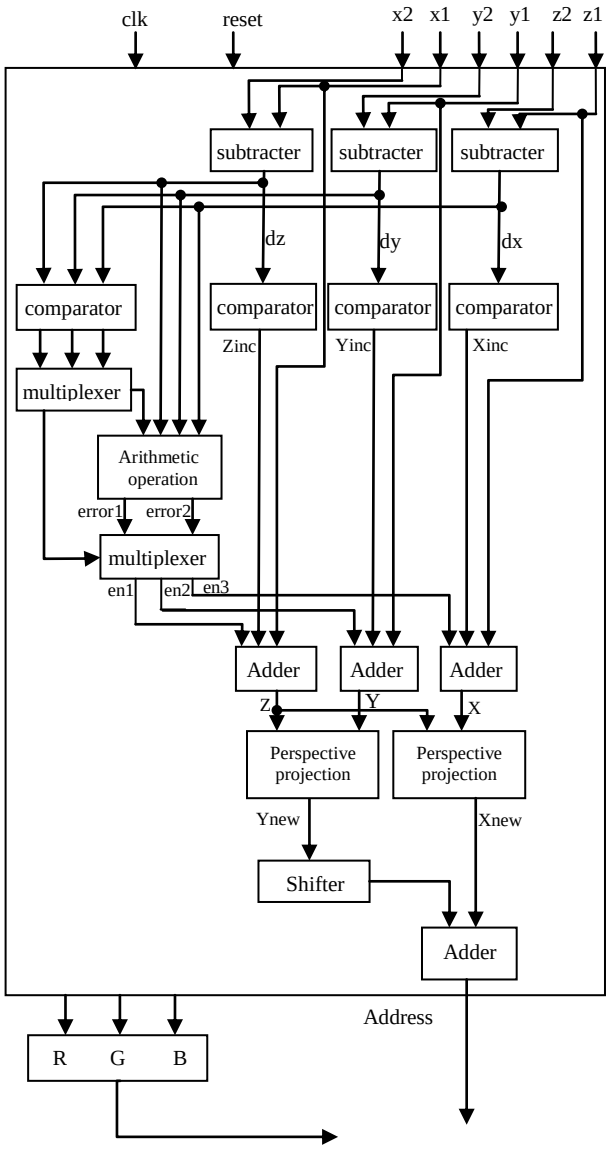

Figure 6: The designed hardware of graphic controller for 3D_Bresenham algorithm

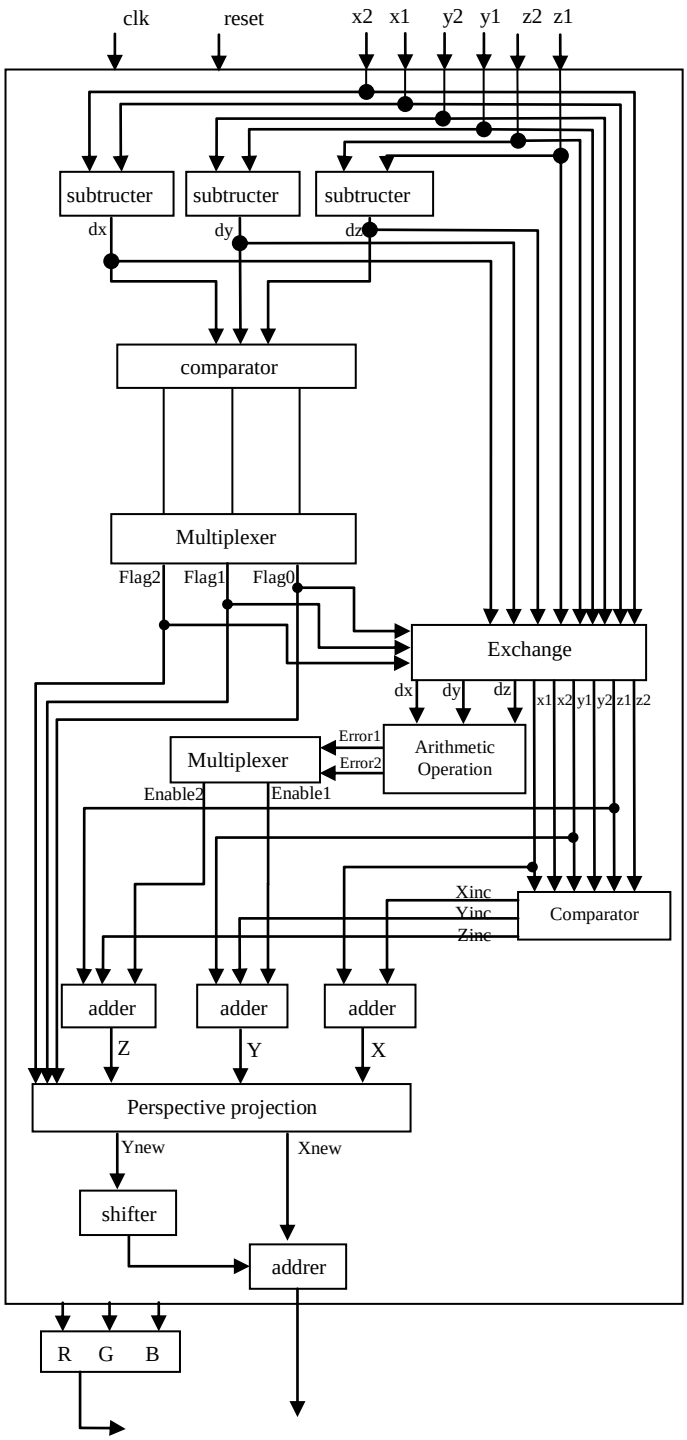

Figure 7: The designed hardware of graphic controller for Modified 3D_Bresenham algorithm

Tikrit Journal of Engineering Sciences/Vol.20/No.2/March 2013, (37-47)

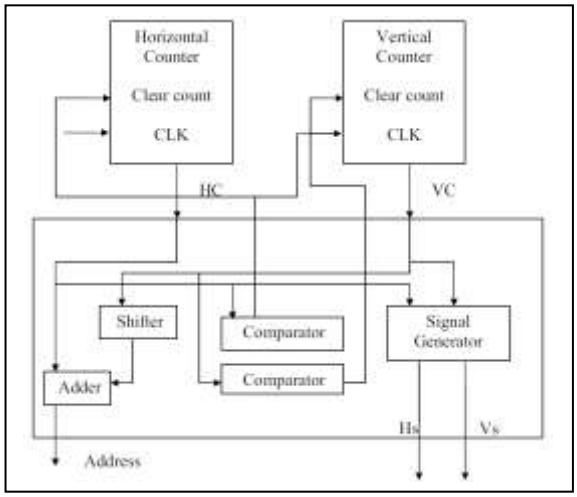

Figure 8: Designed Hardware Refresh Controller

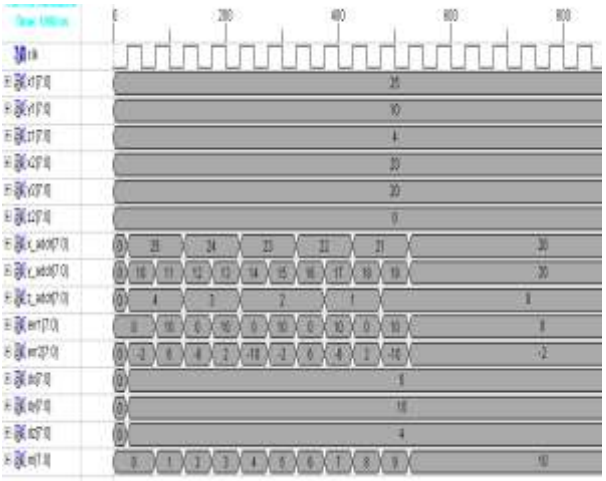

Figure 9: Simulation sample results of example 1 and example 2 using VHDL

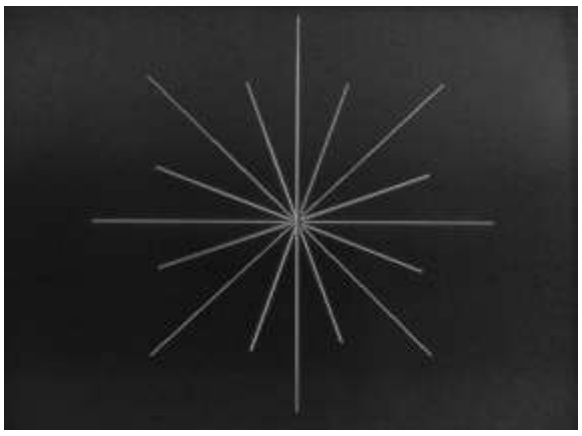

Figure 10: Hardware unit results using FPGA tha displays a Series of 3D_ Lines using the 3D _ Bresenham's algorithm and the modified algorith

Table 1: Results of Example 1 and Example 2 Using OpenGL.

Counter	x	y	z	err1	err2
10	25	10	4	0	-2
9	25	11	4	10	6
8	24	12	3	0	-6
7	24	13	3	10	2
6	23	14	2	0	-10
5	23	15	2	10	-2
4	22	16	2	0	6
3	22	17	1	10	-6
2	21	18	1	0	2
1	21	19	0	10	-10
0	20	20	0	0	-2

Table 2: Resources utilization for the 3D- Bresenham algorithm unit

Type Resources	Used Resources	Total Resources	Ratio
Number of Slices	397	4656	8%
Number of Slices Flip Flops	548	9312	5%
Number of 4 input LUTs	313	9312	3%
Number of Bounded IOBs	7	232	3%
Number of Block RAMS	19	20	95%
Number of MULT18X18SIOs	2	20	10%
Number of GCLKs	2	24	8%
Maximum Operating Frequency	76.196 MHz		
Minimum period	13.124 ns		

Table 3: Resources utilization for the 3D- modified Bresenham algorithm unit

Type Resources	Used Resources	Total Resources	Ratio
Number of Slices	369	4656	7%
Number of Slices Flip Flops	541	9312	5%
Number of 4 input LUTs	266	9312	2%
Number of Bounded IOBs	7	232	3%
Number of Block RAMS	19	20	95%
Number of MULT18X18SIOs	2	20	10%
Number of GCLKs	2	24	8%
Maximum Operating Frequency	68.334MHz		
Minimum period	14.634ns		