NTU Journal of Engineering and Technology (2023) 2 (1) : 6-15
DOI: https://doi.org/10.56286/ntujet.v2i1

IRAQI Academic Scientific Journals

P-ISSN: 2788-9971 E-ISSN: 2788-998X
NTU Journal of Engineering and Technology
Available online at: https://journals.ntu.edu.iq/index.php/NTU-JET/index

Graphics Accelerators: A Review

1st Layla Jamal Hussein[1], 2nd Basma MohammadKamal Younis[2], 3rd Ahmed Khazal Younis[3]
1. M.tech. Student, Department of Computer Technology Engineering, Engineering Technical College, Northern Technical University, Iraq, 2. Department of Computer Technology Engineering, Engineering Technical College, Northern Technical University, Iraq, 3. Department of Computer Technology Engineering, Engineering Technical College, Northern Technical University, Iraq

Article Informations

Received: 16-12- 2022,
Accepted: 09-01-2023,
Published online: 15-03-2023

Corresponding author:
Name: Layla Jamal Hussein
Affiliation : Engineering Technical College ,Northern Technical University
Email: layla.jamal@ntu.edu.iq

Key Words:
Graphics Accelerator,
Computer Graphics Algorithms,
HLS,
FPGA,
GPU.

A B S T R A C T

The spreadability of large and diverse computer graphics applications, highly powerful and programmable hardware platforms, rapid advances in their programing techniques have all permitted design different hardware accelerators for many applications. While the rapid increase in “graphics performance accelerators “have made it a compelling platform for computational requirements tasks in several applications areas. Therefore, graphics accelerators have attracted us because of their important applications. In this paper tries to reach those objectives through a regular review of similar studies within this field of research. The process started by intensely surfing the famous specialized digital libraries, as a result, “40 related works “were collected and examined upon several important technical subjects: technical motivations that underlie the chosen graphics algorithm, the software development that led to interest in graphic algorithms, hardware platform used , programming tool, and the design intended for 2D or 3D graphics. We believe that the software techniques presented in this paper will be helpful for researchers who plan to develop new “graphics accelerators algorithms “. Also, it gives the required introductions in developing “hardware graphics accelerators “and help select materials, techniques, and tools.

1. Introduction

Graphics accelerators are hardware components optimized for doing the calculations for 2D or 3D computer graphics[1]. The graphic accelerator controls the display system and supports all processes and functions needed. Some of them can support multiple display devices [2] . GPUs also can be used to display stereoscopic imagery, where two images are calculated to show them to the corresponding eye. The stereoscopic graphics accelerator needs distinct capabilities more than a traditional one [3].

Graphics algorithms are characterized by the complexity of dealing with a massive amount of data on hand, and their need for high-level computing so distinctive processors for computer graphics are designed with certain accelerators for each specific algorithm [4], [5]

Today, Computer graphics accelerators chips are may be the most powerful computational hardware [6], [7]. These chips are known as Graphics Processing Units (GPUs). It moved from supplementary parties to accident , strong , and programmable processors . Many researchers and designers have become interested in using graphics hardware for general purpose. In latest years, there has been an increase in interest in such research endeavors, termed GPGPU (General Purpose GPU Computing) [5], [7].

The purpose of the current study is to review a group of graphics accelerators that focuses on hardware ones and their various applications, whether for computer graphics, image processing, or even for general uses. This paper composes six sections; the first describes the motivation to present this review, the second involves the theoretical aspects of the basic graphics algorithms used in the gathered papers, the third illustrates the hardware platform used. Also, it includes a practical part for each architecture, section four presents the experimental results and analysis of the performance obtained through the methods reviewed in the previous sections. Finally, section five contains conclusions.

2. Theory

Computer graphics are known as pictorial representations or graphical representations of objects. In computer graphics, a bitmap display system is used to generate and display images [8]. The diverse applications of computer graphics hardware are achieved by rapid optimization in programmability and performance for vital elements of graphics hardware [3], [9]. This section will expla the development of computer graphics algorithms and describe their basics.

2.1 Graphics Pipeline

Graphics pipeline are a basic concept in computer graphics conceptually organized as several stages in a pipeline way through which data and commands that describe the scene pass when presented. The term rendering pipeline is also used. These stages are the application, geometry and raster stages [10]- [12]. They are presented in figure 1.

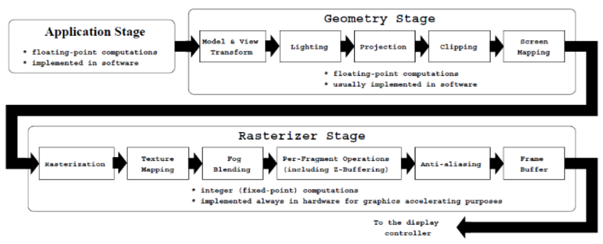
Application Stage
* floating-point computations
* implemented in software
Geometry Stage
Model & View Transform
Lighting
Projection
Clipping
Screen Mapping
* floating-point computations
* usually implemented in software
Rasterizer Stage
Rasterization
Texture Mapping
Fog Blending
Per-Fragment Operations (including Z-Buffering)
Anti-aliasing
Frame Buffer
* integer (fixed-point) computations
* implemented always in hardware for graphics accelerating purposes
To the display controller

Figure 1: A typical 3D graphics pipeline

The application stage only deals with the object specification directly as the objects are created from connected geometric shapes and specifications to appropriate another stage. Figure 1 illustrates these stages implemented in software [10], [12].

The geometric stage: is the stage responsible for most operations of the per-primitive or per-vertex. Basically, in this stage, matrix transformations are applied to the primers received from the application stage, resulting in the mapping of the input polygons to the 2D display. This stage has many functional substages: the model and transforms, lighting, projection, clipping, and the screen mapping stages. In the model and transform stage, the coordinates are converted from input model coordinates to a common system coordinate. In the lighting stage, light sources and properties of the material are calculated. In the projection stage, a perspective type is used to transform 3D objects into a 2D plane, from which we are viewing. The clipping stage removes objects that will not be visible from the scene this operation can be done either by discarding objects that are outside of the field of view, or too far a way, or clipping objects that intersect with any clipping plane. Finally, in the screen mapping stage, the scene data is transformed to its equivalent screen coordinates. The geometric stage has usually implemented in software, although there are high-performance graphical systems that implement this conceptual stage in hardware. At this stage floating-point calculations are performed [12].

The goal of the rasterizer stage is to assign correct colors to the pixels on the screen to render an image correctly. This process is called rasterisation or scan conversion. Unlike the geometry stage, which handles per-primitive operations, the rasterizer stage handles per-pixel operations. During rasterisation, the data represented by the object database (colors and texture) is transformed into a pixel-based image. A scan-line algorithm is commonly used to give the final image. The pixel has scanned a line at a time and their color is determined from the polygons that contribute to that part of the screen. The 2D images of the projected primitives are stored in a memory called the frame buffer, which is read periodically by the display controller to form the image on the screen. The rasterisation is concerned only with the production of a series of frame buffer addresses and values known as (fragments). Each fragment is fed to the "depth-buffer" or "Z-buffer "to solve the visibility problem. Because of the sampling process involved by rasterisation, fragment values may suffer from an aliasing phenomenon (e.g., the stair-case effect of lines drawn on a raster screen), this will be overcome in the next process which is called "anti-aliasing ". Finally, the surviving part of the frame buffer, after all the primers have been processed, will produce the final image. The rasterizer stage is implemented in "hardware" wherever exists the need for "graphics acceleration" and it usually involves only integer arithmetic [12].

2.2 Graphics Algorithm

The references are carefully studied. Figure 2 illustrates the percentages of using graphical accelerators in various applications; for computer graphics, image processing, or general applications. The main algorithms used in the design of graphics accelerators will be presented in this section. Figure 3, depicts the percentage use of these algorithms in hardware accelerators design.

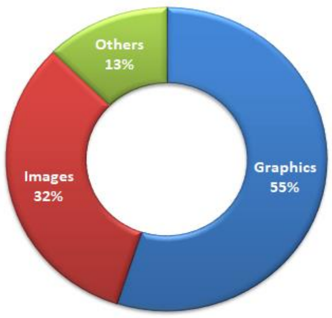

Figure 2. Proportion use of computer graphics accelerator applications for all the reviewed studies.

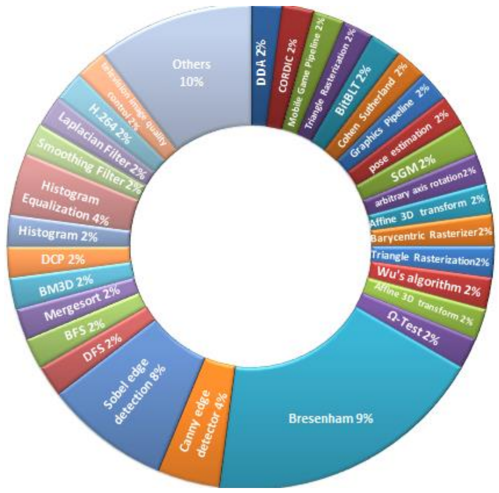

Figure 3: Proportion use of computer graphics algorithm in hardware design for all the reviewed studies.

2.2.1 Line Generation Algorithms

This algorithm is for drawing a line on discrete graphics devices. It rasterizes lines in one color [8], [13], [14]. Improving representation with multiple degrees of color requires an advanced process, spatially anti-aliasing [1]. The most famous line algorithms are DDA [14] and Bresenham [8], [15], both of them used for 2D or 3D [16], [3], but the latter is used in designing graphics accelerators because it uses integer calculations without affecting the accuracy of line drawing. Many researchers have either used [3], [13] or developed [8], [17] Bresenham's algorithm. The development is either to work in 3D or be implemented using FSM [18] or remove aliasing by integrating it with Wu's algorithm [1]. The flowchart for this algorithm in its simple 2D form can find in [1], 3D in [8].

2.2.2 Circle Generation Algorithms

A circle can be created using two algorithms: Bresenham, and the midpoint circle algorithms. The Bresenham 2D algorithm is used to draw lines to generate the circuit by assigning parameters at each sampling step. Square roots are required to calculate distances in pixels. The Bresenham circle algorithm avoids square root calculations by comparing square distances to pixels. Therefore, the Bresenham's algorithm is used in the design of graphics accelerators [2], [19].

2.2.3 Clipping Algorithms

Removing objects that will not be visible from the scene can perform this operation by clipping objects that intersect with any cutting level. There are two kinds of clipping (Hardware clipping and Software clipping). The hardware clipping machine is a two-dimensional automatic clipping machine. The rectangular clipping screen can be defined by four-piece records where each record is loaded with the value of the border it represents. A software clipping is essential for two reasons. The first is to clip those polygons that are partly in the positive space and partly in the negative space and cannot be properly handled by the hardware cutting machine, and the second reason for using software clipping is that it is necessary to prevent the projected header values of any polygon from exceeding the capacity of the devices or the acceptable range of device coordinates. The most commonly used algorithm in the design of graphics accelerators is Cohen Sutherland [20], [21].

2.2.4 Transformation Algorithms

Usually, two main categories of transformation are used in graphics systems. Coordinate transformations deal with the shift from one coordinate system to another. In the second order, geometric transformations are applied to entities within a single coordinate system. Many graphics applications include a series of geometric transformations. Animations, e.g., may require translating and rotating an object. So, geometric transformations are the most commonly used algorithms in the design of graphics accelerators [22], [23], [7], [9].

2.2.5 Modeling Algorithms

One of the major concepts in computer graphics is the modeling of objects and images. This means describing objects and images to produce a visual presentation [24]. One way to do this is to use a set of primitive or geometric shapes to execute on a computer but flexible to represent or model a variety of objects. This technique describes a 3D object as a set of surfaces that separate the internal object from the environment. Typical examples of boundary representations are polygon faces [25].

2.2.6 Projection Algorithms

Projection is the stage in which an object transforms from a three-dimensional space to its image in a two-dimensional space, because the surface of the display is only two-dimensional [3], [26]. This provides the viewer with a depth signal. Therefore, this type of projection is used in most hardware accelerators [3], [13].

2.2.7 Hidden Surface Detection Algorithms

One of the most common problems in computer graphics is the removal of hidden parts of body images. All parts of the object are displayed, including many parts that must be invisible. To remove these parts, you must apply a hidden line or hidden surface algorithm. The algorithm works on different types of scene models, generating different output forms [27]. Among these algorithms, the Z-Buffer algorithm is used to solve the visibility problem and is also used in the design of hardware graphics accelerators. The Z-buffer is a dedicated piece of memory that is divided into 16, 24 or 32-bit words, each of which is linked to a different pixel on the screen. The Z-buffer store is used to store the depth of the 3D point that is displayed on the corresponding pixel. A density buffer is used to store the density of this pixel in each pixel position. The frame buffer can be used to display objects and remove hidden objects by scanning all faces and updating the pixel value only when the depth of the point that appears on the pixel is less than the depth stored in the pixel z buffer. The front faces of the object can be converted into an image in any order [12], [13].

3. Implementation of graphics hardware accelerators

Table-1 in appendix illustrates a summary of the collected data from " 40 reviewed works " [1]-[3], [5]-[11], [13], [14],[16]-[21], [23], [26], [28]-[47]. Following sections describe the details results obtained from these researches that we think are useful for designers work in this field.

3.1 Hardware platforms

The hardware platforms are several of a compatible device on which software applications can excudit. Each hardware platform has its own program language, so a special platform software should be created that includes a standard type of processor and hardware parts associated with it [48], [49]. After reading the researches under study from hardware platforms used. Figure 4 illustrates these platforms used in the design of graphics accelerators. This figure shows the dominance of FPGAs then GPUs.

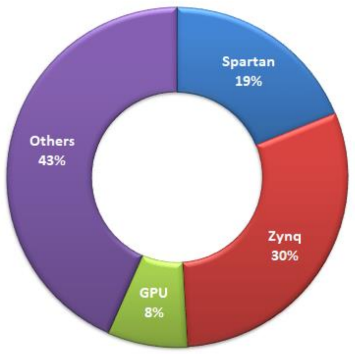

Figure 4: Proportion use of hardware platforms used in graphics accelerator for all the reviewed studies.

3.2 Software design tools

The software environment is used to write and run applications. These include software tools such as graphical interface builders, pool creators, class libraries, and application development utilities. Figure 5 shows the software platforms used in the design of graphics accelerators, which illustrates the dominance of VHDL and HLS on these platforms.

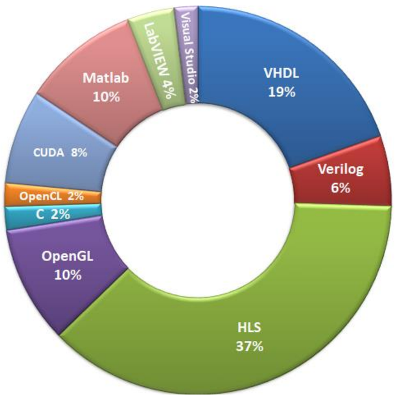

Figure 5: Proportion use of Software platforms used in graphics accelerator design for all the reviewed studies.

4. Results and analysis

The graphics accelerators of the reviewed references have been built according to the researcher's need, either to speed up a graphics algorithm, hardware system, to fit the space of the hardware platform used in the design, or to reduce the consumed power as shown in Figure 6. Speed is the dominant measure used by researchers during design. By studying references, the improved speed increased speed of performance, as high speed, scalability and efficiency can be achieved using FPGA devices compared to other implementation-based computing mechanisms [50]- [52]. Designers may increase performance, power, or cost to reduce design time [53]- [55]. It occupies 74% of the total proportion of the research under study. Power consumption is a major factor for battery-based systems [55], [56]. Results of the power improvement, are also presented by studying the references in which we observed a significant reduction in power consumption while maintaining system performance. This measurement tool occupies 9% of the total proportion of the research under study. The results from the references study showed how improvement guidelines applied to the timing and improvement of the area. Resources can be allocated as needed (which is subject to the area or energy) [57]. It occupies 17% of the total proportion of the research under study.

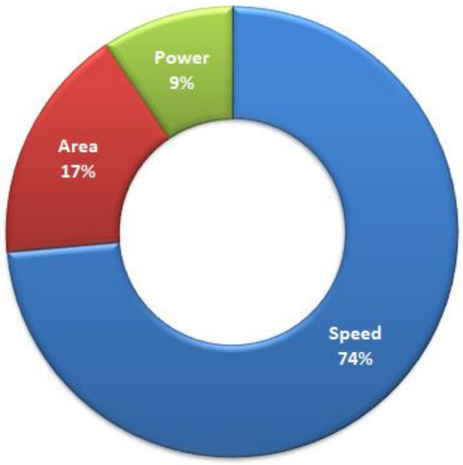

Figure 6: Proportion main purpose used in graphics accelerator design for all the reviewed studies.

5. Conclusion

We provided an extensive survey and classification to develop new algorithms for graphics accelerators design. A summary was provide in the development of hardware graphics accelerators. We then described the improvements that modern research focuses on. The purpose of reviewing articles published in the past 10 years is to help choose technology and tools in the development of these accelerators. Figure 7 in the appendix show the flowchart for the system.

References

[1] B. M. Younis and A. K. Younis, "Hardware accelerator for anti-aliasing Wu's line algorithm using FPGA," Telkomnika, vol. 19, no. 2, pp. 672–682, 2021.
[2] A. H. Ali and R. Z. Mahmood, "Bresenham's Line and Circle Drawing Algorithm using FPGA," AL-Rafidain J. Comput. Sci. Math., vol. 15, no. 2, pp. 39–53, 2021.
[3] M. R. Ahmed and B. M. Younis, "3D Stereo Rendering Using FPGA," Comput. Eng. Intell. Syst., vol. 10, no. 3, p. 19, 2019.
[4] J. Vince and J. A. Vince, Mathematics for computer graphics, vol. 3. Springer, 2010.
[5] A. M. Ibrahim and S. E. D. Habib, "Design and Implementation Of Cairo University Graphics Processing (CUGPU) Using HLS Approach," in 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2021, pp. 1–3.
[6] P. Renc, T. Pęcak, A. De Rango, W. Spataro, G. Mendicino, and J. Wąs, "Towards efficient GPGPU Cellular Automata model implementation using persistent active cells," J. Comput. Sci., p. 101538, 2022.
[7] S. Nawfal and F. Ali, "The acceleration of 3D graphics transformations based on CUDA," J. Eng. Des. Technol., 2018.
[8] B. M. K. Younis and L. N. S. M. Sheet, "Hardware Implementation of 3D-Bresenham's

Algorithm Using FPGA," Tikrit J. Eng. Sci., vol. 20, no. 2, pp. 37–47, 2013.
[9] S. N. Alrawy and F. H. Ali, "GPU Accelerated Rotation About an Arbitrary Axis," in 2018 International Conference on Advanced Science and Engineering (ICOASE), 2018, pp. 36–41.
[10] V. Kasik and A. Kurecka, "FPGA Implementation of a Simple 3D Graphics Pipeline," Adv. Electr. Electron. Eng., vol. 13, no. 1, pp. 39–47, 2015.
[11] D. Corbalan-Navarro, J. L. Aragon, M. Anglada, E. De Lucas, J.-M. Parcerisa, and A. Gonzalez, "Omega-Test: A Predictive Early-Z Culling to Improve the Graphics Pipeline Energy-Efficiency," IEEE Trans. Vis. Comput. Graph., 2021.
[12] D. Crisu, S. Cotofana, and S. Vassiliadis, "A hardware/software co-simulation environment for graphics accelerator development in ARM-based SOCs," 2002.
[13] B. M. Kamal and N. Salim, "A real time dynamic 3D graphics processor using FPGA," Int. J. Res. Dev. Eng. IJRDE, vol. 2, no. 1, pp. 1–12, 2013.
[14] F. Hamid Ali, "Depth Buffer DDA Based on FPGA," Al-Rafidain Eng. J. AREJ, vol. 19, no. 5, pp. 28–39, 2011.
[15] M. Daniel, "A Mathematical Overview of Bresenham Algorithms in the Determination of Active Pixel Positions".
[16] C. Desmouliers, E. Oruklu, S. Aslan, J. Saniie, and F. M. Vallina, "Image and video processing platform for field programmable gate arrays using a high-level synthesis," IET Comput. Digit. Tech., vol. 6, no. 6, pp. 414–425, 2012.
[17] S. Ismae, O. Tareq, and Y. T. Qassim, "Hardware/software co-design for a parallel three-dimensional bresenham's algorithm.," Int. J. Electr. Comput. Eng. 2088-8708, vol. 9, no. 1, 2019.
[18] K. Panek, B. Flak, S. Koryciak, and K. Wiatr, "Basic 3D graphics processor implemented on small FPGA," Meas. Autom. Monit., vol. 64, 2018.
[19] J. K. Kim, J. H. Oh, J. H. Yang, and S. E. Lee, "2D Line draw hardware accelerator for tiny embedded processor in consumer electronics," in 2019 IEEE International Conference on Consumer Electronics (ICCE), 2019, pp. 1–2.
[20] A. E. Beasley, C. T. Clarke, and R. J. Watson, "An OpenGL Compliant Hardware Implementation of a Graphic Processing Unit Using Field Programmable Gate Array–System on Chip Technology," ACM Trans. Reconfigurable Technol. Syst. TRETS, vol. 14, no. 1, pp. 1–24, 2020.
[21] A. I. Dawod, "Hardware Implementation Of Line Clipping A lgorithm By Using FPGA," Tikrit J. Eng. Sci., vol. 18, no. 3, pp. 89–105, 2011.
[22] F. H Ali and A. I Dawod, "FPGA Based Implementation Of Concatenation Matrix," Al-Rafidain Eng. J. AREJ, vol. 18, no. 2, pp. 15–31, 2010.
[23] D. Ali and H. Fakhrulddin, "Transformation matrix for 3D computer graphics based on FPGA," Al-Rafidain Eng. J. AREJ, vol. 20, no. 5, pp. 1–15, 2012.
[24] S. Dhar and S. Pal, "Surface Reconstruction: Roles in the Field of Computer Vision and Computer Graphics," Int. J. Image Graph., vol. 22, no. 01, p. 2250008, 2022.
[25] J. Romero, D. Tzionas, and M. J. Black, "Embodied hands: Modeling and capturing hands and bodies together," ArXiv Prepr. ArXiv220102610, 2022.
[26] F. Jabar, J. Ascenso, and M. P. Queluz, "Perceptual analysis of perspective projection for viewport rendering in 360° images," in 2017 IEEE International Symposium on Multimedia (ISM), 2017, pp. 53–60.
[27] L. M. Saeed and F. H. Ali, "Evaluation of Hidden Surface Removal Methods Using Open GL," Al-Rafidain Eng. J. AREJ, vol. 26, no. 2, pp. 300–308, 2021.
[28] K. Georgopoulos et al., "An evaluation of vivado HLS for efficient system design," in 2016 International Symposium ELMAR, 2016, pp. 195–199.
[29] H. M. Abdelgawad, M. Safar, and A. M. Wahba, "High level synthesis of canny edge detection algorithm on Zynq platform," Int J Comput Electr Autom Control Inf Eng, vol. 9, no. 1, pp. 148–152, 2015.
[30] B. Vaidya, M. Surti, P. Vaghasiya, J. Bordiya, and J. Jain, "Hardware acceleration of image processing algorithms using Vivado high level synthesis tool," in 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), 2017, pp. 29–34.
[31] K. S. Ay and A. Doan, "Hardware/software co-design of a 2d graphics system on fpga," Int. J. Embed. Syst. Appl. IJESA, vol. 3, no. 1, 2013.
[32] A. B. Amara, E. Pissaloux, and M. Atri, "Sobel edge detection system design and integration on an FPGA based HD video streaming architecture," in 2016 11th International Design & Test Symposium (IDT), 2016, pp. 160–164.
[33] N. A. M. Daud, F. Mahmud, and M. H. Jabbar, "A hardware acceleration based on high-level synthesis approach for glucose-insulin analysis," in AIP Conference Proceedings, 2017, vol. 1788, no. 1, p. 030087.
[34] R. Kumar, B. K. Kaushik, and R. Balasubramanian, "FPGA implementation of image dehazing algorithm for real time applications," in Applications of digital image processing XL, 2017, vol. 10396, p. 1039633.
[35] A. Qamar, F. B. Muslim, F. Gregoretti, L. Lavagno, and M. T. Lazarescu, "High-level synthesis for semi-global matching: Is the juice worth the squeeze?," IEEE Access, vol. 5, pp. 8419–8432, 2016.
[36] Y. Zheng, "The design of sobel edge extraction system on FPGA," in ITM Web of Conferences, 2017, vol. 11, p. 08001.
[37] H. W. Oh, J. K. Kim, G. B. Hwang, and S. E. Lee, "The Design of a 2D Graphics Accelerator for Embedded Systems," Electronics, vol. 10, no. 4, p. 469, 2021.
[38] R. He and Z. Chen, "Design of Image transmission and Display System Based on ZYNQ," 2018.

[39] R. Ghodhbani, L. Horrigue, T. Saidani, and M. Atri, “Fast FPGA prototyping based real-time image and video processing with high-level synthesis,” Int J Adv Comput Sci Appl, vol. 2, pp. 108–116, 2020.
[40] K. F. Lysakov, K. K. Oblaukhov, and M. Y. Shadrin, “Implementation of FPGA algorithms for identification of image distortion due to compression,” Optoelectron. Instrum. Data Process., vol. 56, no. 1, pp. 28–32, 2020.
[41] F. Guo, W. Wan, W. Zhang, and X. Feng, “Research of graphics acceleration based on embedded system,” in 2012 International Conference on Audio, Language and Image Processing, 2012, pp. 1120–1124.
[42] J. Monson, M. Wirthlin, and B. L. Hutchings, “Optimization techniques for a high level synthesis implementation of the Sobel filter,” in 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2013, pp. 1–6.
[43] D. Wang, J. Xu, and K. Xu, “An FPGA-based hardware accelerator for real-time block-matching and 3D filtering,” IEEE Access, vol. 8, pp. 121987–121998, 2020.
[44] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, “Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis,” IEEE Access, vol. 5, pp. 2747–2762, 2017.
[45] H. Ishida, H. Furukawa, T. Kyoden, and T. Tanaka, “Development of a wireless power transmission simulator based on finite-difference time-domain using graphics accelerators,” IET Power Electron., vol. 10, no. 14, pp. 1889–1895, 2017.
[46] J. M. Joseph, T. Winker, K. Ehlers, C. Blochwitz, and T. Pionteck, “Hardware-accelerated pose estimation for embedded systems using Vivado HLS,” in 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2016, pp. 1–7.
[47] A. Gorobets and P. Bakhvalov, “Heterogeneous CPU+ GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers,” Comput. Phys. Commun., vol. 271, p. 108231, 2022.
[48] R. Saha, P. P. Banik, and K.-D. Kim, “HLS based approach to develop an implementable HDR algorithm,” Electronics, vol. 7, no. 11, p. 332, 2018.
[49] R. Weber, A. Gothandaraman, R. J. Hinde, and G. D. Peterson, “Comparing hardware accelerators in scientific applications: A case study,” IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 1, pp. 58–68, 2010.
[50] K. AHMED and T. Ercan, “ANFIS Analysis of Wireless Sensor Data with FPGA,” Acta Infologica, vol. 2, no. 1, pp. 22–32, 2018.
[51] T. Ercan and A. K. Al Azzawi, “Design of an FPGA-based intelligent gateway for industrial IoT,” 2019.
[52] P. K. Murthy, “Hardware acceleration of edge detection using HLS,” PhD Thesis, California State University, Northridge, 2019.
[53] D. O’Loughlin, A. Coffey, F. Callaly, D. Lyons, and F. Morgan, “Xilinx vivado high level synthesis: Case studies,” 2014.
[54] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, “High-level synthesis for FPGAs: From prototyping to deployment,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 30, no. 4, pp. 473–491, 2011.
[55] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, and D. Stroobandt, “An overview of today’s high-level synthesis tools,” Des. Autom. Embed. Syst., vol. 16, no. 3, pp. 31–51, 2012.
[56] R. Ben Atitallah, N. Fakhfakh, and J.-L. Dekeyser, “Exploring HLS Optimizations for Efficient Stereo Matching Hardware Implementation,” 2017.
[57] R. Nane et al., “A survey and evaluation of FPGA high-level synthesis tools,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 35, no. 10, pp. 1591–1604, 2015.

Table-1: Summary of the collected data from the 40 reviewed articles.

Ref	Year	Algorithm	Hardware Platform	Software tool	2D/3D	Purpose	Results
[1]	2021	Wu's algorithm	Zynq-7000 XC7Z020	HLS	2D	Speed, Area, Power	Single core area 8% and consumed 1.6W 10 cores area 77% consumed of 2 W. time is 0.31µs for 10 cores equals to tenth of one core time.
[2]	2021	Bresenham's Line, Circle	SPARTAN 3EXC3S500	VHDL	2D	Speed	Refresh rate of 50MHz with resolution 800x600.
[3]	2019	off-axis Bresenham's	Spartan3E XC3S500E	MATLAB VHDL	3D	Speed	Speedup 1266, Operating frequency 35.417MHz Occupied area 85%
[5]	2021	Triangle Rasterization	Vertix7(VC709) , XM105, and ZYBO	HLS, OpenGL	3D	Speed, area	approximately 1.2 M gates area, with rate about 300 K vertices/sec, and the render rate is 13 Mpixels/sec.
[6]	2022	Modeling	Nvidia GPU	CUDA	General	Speed	speeding up simulation times support complex system modeling
[7]	2018	Affine 3D transform	NVIDIAGeForce:1050GTX,610 GT	LabVIEW , Visual Studio, CUDA	3D	Speed	Execution time 0.508 sec for 100 million head conversion using LabVIEW and 0.096 sec using Visual Studio.
[8]	2013	Bresenham	Spartan3EXC3S 1600E	OpenGL VHDL	3D	Speed	speed of 68 M pixels per second Max. Operating Frequency: 68.334MHz
[9]	2018	arbitrary axis rotation	NVIDIAGeForce1050GTX	LabVIEW , CUDA	3D	Speed	Results showed the significant speedup on CUDA/C++ compared to LabVIEW for the same model complexity
[10]	2015	Graphics Pipeline	Spartan-6	VHDL	3D	Speed	real-time rendering 5000 fps
[11]	2021	Ω-Test	ARM Mali-450 GPU	OpenGL	3D	Speed, Area,	savings in GPU/Memory system of 26.4% and speedup of 16.3%
[13]	2013	Bresenham	Spartan3E XC3S500E XC3S1600E	VHDL	3D	Speed	Rendering speed 25.279M Pixel per Second and 169,896 triangles per Second.
[14]	2010	DDA	Spartan-3E	VHDL	3D	Speed	Speed of 120M pixel per second
[16]	2012	Canny edge detector	Xilinx Virtex-5 XC5VLX110T	HLS	Image	Speed	IVPP can be a cost-effective, rapid development and prototyping platform
[17]	2019	Bresenham	Zybo board (Z-7010)	MATLAB VHDL	3D	Speed	Fastest run time achieved is 0.31µs
[18]	2018	Bresenham	Altera NiosII	Verilog	3D	Area	The logical usage was around (50%), and memory usage was 100%
[19]	2019	Bresenham	FPGA	OpenGL, VHDL	2D	Speed	0.6 ms operation time to draw 41 different lines with 166 MHz operation frequency.
[20]	2020	Barycentric Rasterizer	Max10 Stratix V, Cyclone V	OpenGL HDL coder	3D	Speed Power	Pixel processing is 80%. Power consumption conserve 2%
[21]	2011	Cohen-Sutherland	Spartan3EXC3S 200E	VHDL	2D	Speed	designed unit is capable of clipping (232524) line segments per second.
[23]	2011	CORDIC	Spartan3E	VHDL	3D	Speed	Maximum speedup 5 speed of 10 M vertex per second
[26]	2012	Sobel edge detection	Virtex2Pro/Virtex5	HLS	Image	Speed	Using HLS: code is reduced dramatically, you can optimize a design, and Verification times reduced
[28]	2016	DFS, BFS, and Merge sort	Virtex7:XC7VX 415T, XC7VX330T, XC7V690T	HLS	Video image	Area Speed	Parallelizing gives better results and almost reaches FPGA's BRAM, in turn lower clock frequencies.
[29]	2015	Canny edge detection	Zynq AP SoC XC7Z020	HLS	Image	Area Speed	CPU utilization drop down, and frame rate jumps to 60 fps of 1080p

[30]	2017	Histogram, (Equalization, Smoothing Filtering)	ZynqZC602, Virtex7VC709, Virtex6ML605 Spartan6SP605	HLS OpenCV	Image	Speed	ZynqZC602 8.73ns time, frq: 114.5MHz Vertex7VC709 8.18 ns time, frq: 122.2MHz Virtex6ML605 time 8.39ns and 119.2 MHz Spartan6SP605 8.56 ns time, frq: 116.8MHz
[31]	2013	Bresenham BitBLT	VirtexII Pro FPGA	C	2D	Area Speed	Bresenham core: 500pixels in 0.1ms. BitBLT operation: 91 clock cycles per pixel
[32]	2016	Sobel edge detection	ZYBO ZC7010	HLS	Image	Speed	Max Clock Frequency in NvidiaM840 is 110MHz and in Proposed Sobel is 150MHz
[33]	2016	Hovorka (glucose-insulin)	Xilinx Virtex-7	HLS	General	Speed	high-level synthesis beneficial to design an ODE function for HovorkaModel.
[34]	2017	DCP	Zynq ZC702, Zed board	HLS/MATLAB	Image	Speed Area	speed of 29 fps for resolution of 1920x1080 use 9 18K_BRAM, 97 DSP_48, 6508 FFs and 8159LUTs
[35]	2017	SGM	Xilinx ZynqTM 7020	HLS	3D	Speed	performance 30 frames/s for resolution of 640x480 with depth of 128 pixels per frame.
[36]	2017	Sobel Edge Detection	Zed board	HLS	Image	Speed	Speedup of 80
[37]	2021	Bresenham	Xilinx Zynq xc7z010clg400	MATLAB , Verilog	2D	Speed, Area	Operating frequency 100 MHz, Estimated gate account 75,406
[38]	2018	H.264	ZYNQ7000-XC7Z020	HLS	Image	Speed	delay of image transmission and display is 0.28 ns, while open-source library is 0.33 ns
[39]	2020	Histogram Equalization	Zed Board xc7z020clg4842	HLS, MATLAB	Image	Speed	Max frequency: 302MHz for YCbCr to RGB Max frequency: 260MHz for RGB to YCbCr
[40]	2020	Television image quality control	Xilinx Zynq-7000	HLS	Image	Speed	control of video signals at SD-SDI (576i) & HD-SDI (1080i) interfaces
[41]	2012	Mobile Game Pipeline	XUPV5-X110T ML505	Verilog	2D	Speed	Pixels rate 90 M/S when system clock was 100 MHz
[42]	2013	Sobel Edge detection	Zed Board	HLS	Image	Speed	Performance ranged: 10.9 fps to 388 fps.
[43]	2016	BM3D algorithm	Intel’s Arria-10 GX1150	OpenCL HLS	Image	Speed Power	1.2× performance boost and an outstanding 8.3× reduction in energy dissipation
[44]	2017	K-Nearest Neighbor	Virtex-7 690t	HLS	General	Speed Power	The result showed that FPGA is better than GPU for implementation time and energy calculation.
[45]	2017	WPT	Nvidia GTX 1080 GPU	CUDA	General	Speed	simulation results agree with experimental results. largest difference between them was 3 and 8%
[46]	2016	pose estimation	Zed oared	HLS	3D	Speed power	speedup of 1.6, manual tracking of 17 fps, and power is reduced.
[47]	2022	heterogeneous multilevel parallel distributing algorithm	NVIDIA, AMD, Intel GPUs	OpenCL	General	Speed	Parallel performance on super-computers using up to 10,000 cores and multiple GPUs with comparable general performance

Flowchart of the System

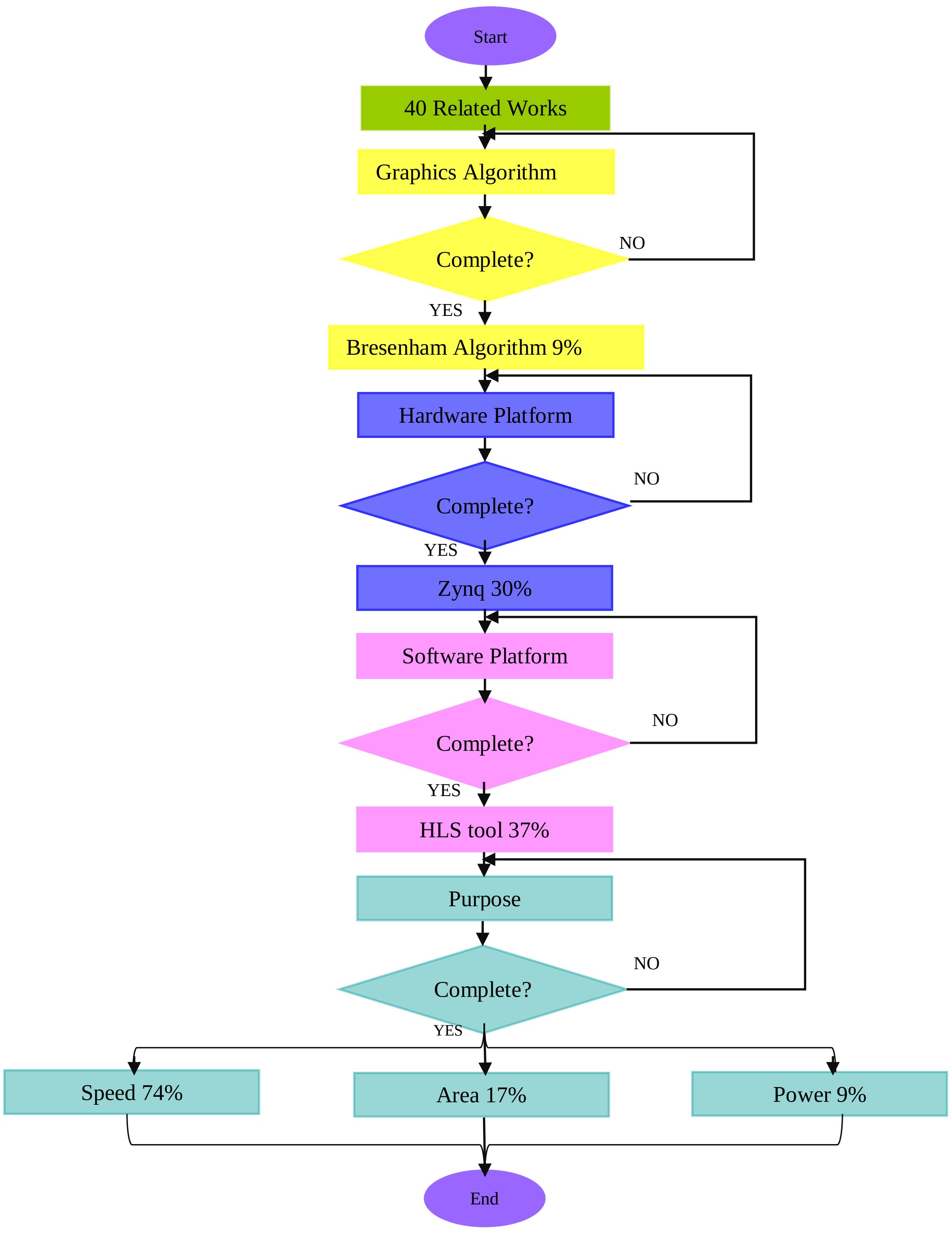

Figure 7: The System Flowchart