Optimization of False-Overlap Detection of Tile Assembly in Tile-based Rendering

Bowen Yang*, Meng Fan†, Mengqiao Han†, Yurong Geng†
*Northwestern Polytechnical University, Xi’ an, China
E-mail:yangbowen@xupt.edu.cn Tel:+86-13991226699
†Xi’an University of Posts and Telecommunications, Xi’an, China
hanmengqiao@163.com,18829290884@163.com,742793340@qq.com

***Abstract*—With the rapid development of mobile devices, terminal devices put forward more requirements on GPU's real-time performance and power consumption. In this paper, tile-based architecture is adopted to improve the utilization of Mobile GPU resources on Mobile Graphics Processor (MGP).This architecture improves the rasterization execution rate through tile-binning generated tile list. However, before the execution of tile-binning, the overlap test will accumulate a large amount of computation when calculating the overlap relationship between primitives and triangles, even if a reasonable detection algorithm is adopted. In this paper, we propose a new method for segmentation of primitives, design an overlap detection algorithm with this method, and analyze the feasibility. Finally, the algorithm is mapped to the tile assembly platform using sort display algorithm, which aims to provide a concise tile list structure and an efficient overlap detection algorithm for mobile graphics processors. By using this algorithm, we can reduce memory overhead by 15% to 35%, CPU latency by 24% to 55%, and computing resources by 32% to 45%.**

Index Terms—primitives segmentation method, overlap detection algorithm, Memory Overhead

I. INTRODUCTION

The use of mobile devices has many limitations, such as finite power supply, low computing power and small size. The most basic problem is power supply. Compared to the development speed of integrated circuits according to Moore's Law [1], battery powered technology is developing much more slowly. Therefore, the low power design of hardware is the focus of research. The key problem to achieve low power consumption is to reduce the internal data transactions between computing component and memory, the processor's annual computing power increases by about 71%, while DRAM bandwidth increases by 25%, which causes the storage wall problem to become more serious [2]. Various mobile graphics hardware approaches have been proposed for optimizing storage performance, including tile-based rendering [3], specialized embedded DRAM [4], compression algorithms and early-z depth test [5]. In tile-based rendering, the storage of tile information is one of the main sources of mobile GPU power consumption [6].

Antochi et al. roughly classified the related display algorithms of the tile-based architecture into three categories [7]: Direct, Two-step, and Sort, where the Sort algorithm is often used because of the highest execution rate. However, the intermediate data structure generated by the Sort algorithm requires the most memory, the solution is to optimize the structure of the tile list, and it can be generally divided into two types: store tiles according to the primitive [8][9][10] and store primitives according to the tile. Among them, the first method requires more storage space and may be better in data reading [11]. So we choice the first method. The key to this method is to detect tiles which do not overlap with primitives.

Although the traditional Bounding box (BBOX) detection algorithm [12][13][14] can easily eliminate tiles that no overlap with primitives, has the advantages of simple design and low hardware utilization. And Hsieh H et al. also proposed a representation method of Bounding box segmentation, but when these methods process some large triangle primitives, the resource utilization and processing efficiency are not high enough .In this paper, based on these methods, an improved approach is proposed to achieve the purpose of eliminating most irrelevant tiles in advance, significantly reducing the cumulative amount of calculation, making the calculation of coverage rate more efficient and accurate.

II. MGP ARCHITECTURE

Since the number of vertex processing is less than the number of pixel processing, which causes vertex processor computing resources to be idle, this will result in performance bottleneck that computing resources cannot be fully utilized [18]. Therefore, this paper uses the unified shader architecture of Fig.1 to use the Z-buffer to solve the performance issue [19].

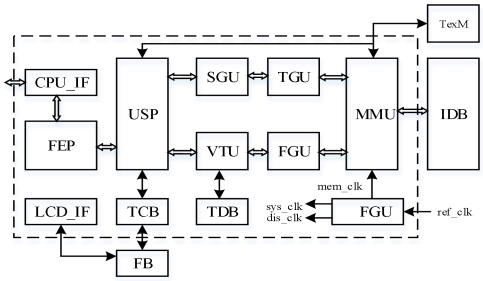


Fig. 1 MGP System Architecture

In the architecture, data, status and control information are exchanged with the main processor through the CPU_IF (CPU Interface) module, and signal synchronization between bus data and graphics processing hardware is completed. Then, the FEP((Front End Processor) unit is used to decompose, organize and preliminarily schedule the graphics rendering tasks.And completing the transformation and illumination processing of the vertexes in the USP (Unified Shading Processor), including the vertexes, texture coordinates, raster position, light source position and illumination calculation of the vertex coordinates, etc. Finally, the SGU (Screen-coordinate Generating Unit) performs operations such as element assembly, cropping, and viewport transformation, and the tile is divided by the TGU unit, in order to create a tile list, which send to the IDB (Intermediate Data Buffer)for storage through the MMU (Memory Management Unit).

In the pixel processing stage, perform rasterization and anti-aliasing on the FGU (Fragment Generating Unit) tile by tile, output to the VTU (Visibility Test Unit) pixel by pixel and compare the depth of each pixel in the TDB (Tile Depth Buffer) to eliminate hidden surfaces. Then send it to the USP unit to execute lighting calculation, texture mapping, etc. The pixel color is saved in the TCB (Tile Color Buffer). After processing the individual tile, write its pixel coordinate position and attributes to the FB (Frame buffer).

III. RELATED WORK

In order to reduce the complexity of the assembly process of the primitive, the data of the vertexes of the primitive are uniformly represented by triangles, points and lines are converted into triangles, thereby unifying the data to simplify data structure and the processing of the rasterization stage.

Based on the Open GL ES2.0 standard [20], the original vertex data types are divided into three categories: points, lines, and triangles. The lines include: L_LOOP, L_STRIP, and LS. The triangles include: TR_STRIP, TR_FAN, and TRS. It is assumed that a series of vertex data V0, V1, V2, V3, V4, V5, V6 etc.ares input. The line and the triangle indicate the vertex data form as shown in Fig.2 and Fig.3 respectively.

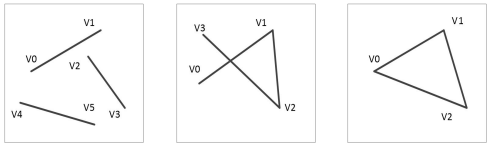


Fig.2 L_LOOP, L_STRIP and LS three line types

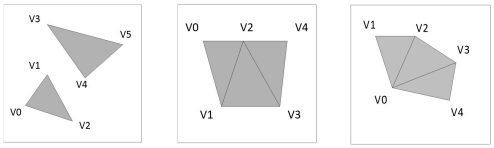


Fig.3 TRS, TR_STRIP and TR_FAN triangle types

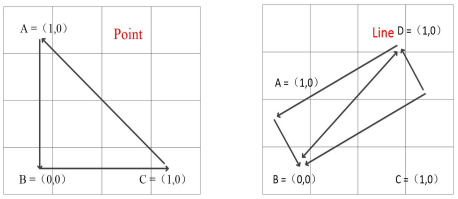


Fig.4 Triangle representation of points and lines

Fig.4 illustrates the method of converting points and lines into triangle representations.Fig.4 shows the representation of the three triangles. How to choose the mode requires consideration of the compression ratio. The graphics vertex compression ratio of TRS is the smallest, using 3 vertex data to represent 1 triangle primitive, but TR_STRIP and TR_FAN types using 1 vertex data to represent 1 triangle primitive. Therefore, in order to obtain a better graphics vertex data compression ratio, most of the TR_STRIP and TR_FAN triangle types are used. This article uses the triangle representation of TR_STRIP. Fig.5 illustrates the conversion of vertex data from the TRS to TR_STRIP process and the recording process of the mask.

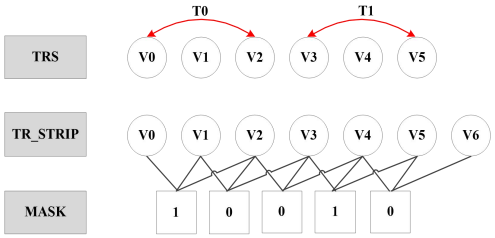


Fig.5 Conversion process from TRS to TR_STRIP

In summary, the basic data conversion process of the vertex data stream is as follows: First, each of the three vertexes in the data stream is divided into a group, and each group can represent a triangle primitive, which is combined into TRS form. Secondly, the TRS is converted in the form of TR_STRIP and recorded with a mask.

IV. PRIMITIVE REPRESENTATION METHOD AND OVERLAP DETECTION ALGORITHM

A. Traditional primitive representation and its limitations

In the traditional detection algorithm, the triangle primitive is usually represented by BBOX. According to the maximum value of the three vertexes of the triangle, a BBOX containing the triangle primitive is made, and the detection algorithm only needs to process the tile inside the BBOX. As shown in Fig. 6, the orange area intersects the bounding box ABCD, and all the tiles in the area are processed when the detection is performed. The gray area does not intersect the BBOX, so the triangle primitive have no coverage relationship with all tiles in this area.

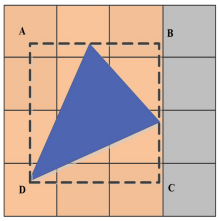


Fig.6 The BBOX represent the triangle primitive

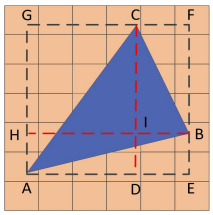
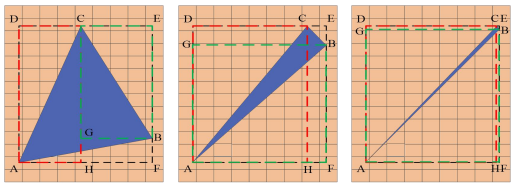


Fig.7 Representation method of dividing triangle primitive BBOX

Hsieh H et al. designed a segmented BBOX representation method. They can divide the BBOX of a triangle primitive into three rectangular boxes according to the original vertex of the triangle, and these vertexes are also the vertexes of the BBOX. Then, the edges of the triangle primitives become the diagonal of these rectangles, and coverage detection becomes a mathematical model processing. As shown in Fig. 7, the triangle element bounding box AEGF is divided into three rectangular boxes: ADCG, IBFC and AEBH. The calculation of the correlation detection algorithm is performed in each rectangular box. The advantage of this approach is that it can be in three processed in parallel within three rectangular boxes to speed up the processing.

Although the segmented triangle primitive BBOX representation method can divide a bounding box into multiple rectangular boxes for processing, it can reduce the number of tiles in BBOX, and improve the computational parallelism. For larger triangular primitive and some special smaller triangle primitive, there are still many tiles which need to be calculated, resulting in excessive computational cumulant and low processing efficiency. These two cases are discussed separately below.

For the larger triangular primitives shown in Fig.8 (a), the segmented BBOX representation method can divide the bounding box AFED into multiple parallel processing rectangles, but some rectangular boxes still contain a large number of tiles. For example, there are still more tiles in the red rectangular box AHCD and the green rectangular box BECG, which need to perform overlap detection calculation, and then accumulate the calculation amount. If the two vertexes C and B of the triangle in Fig.8(a) are moved to the right and up respectively, then the case shown in Fig. 8(b) will be obtained. In this case, the size of the bounding box AFED is constant, the area of the triangle ABC is reduced, which leads to a sudden increase in the number of tiles in the red rectangular frame AHCD and the green rectangular frame AFBG in which the overlap detection calculation is required, and even if a sufficiently optimized detection algorithm is used, a huge amount of calculation is accumulated. In rasterization, narrow-angle triangles are often used to test the accuracy of algorithms such as attribute differences. The characteristics of such triangles are fine and long, as shown in Fig. 8(c), when the two vertexes C and B of the triangle are very close to the point E, a long and narrow triangle appears, and the area of the triangle ABC is very small, so that the number of tiles in the red rectangle box AHCD and the green rectangle box AFBG that need to be overlapped and detected is almost equal to the number of tiles in the AFED, in which case the cumulative amount of calculation is the largest.



a. Larger area triangle primitive b. Smaller area triangle primitive c. Long narrow triangle primitive

Fig.8 Larger triangle element segmentation method

The right triangle element shown in Fig.9, although the triangle element does not look very large (the number of Tiles contained in the bounding box is much smaller than that shown in Fig. 7), since the triangle vertex ABC and the bounding box vertex ABDC are coincident, the bounding box cannot be divided into multiple rectangular boxes. So there is still a part of the Tile that needs to perform overlap detection calculation.

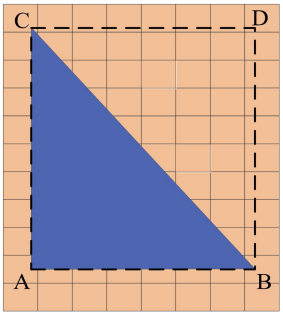
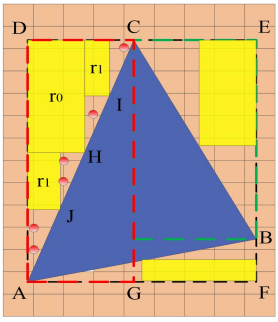


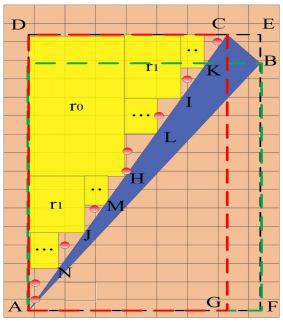
Fig.9 Right triangle primitive

B. *Improved primitive representation*

In view of the above problems, this section proposes an improved triangle element segmentation method, which can eliminate the unrelated Tiles in advance and significantly reduces the computation.Based on this, the detection algorithm is proposed. As shown in Fig.10(a), taking the red rectangular frame AGCD as an example, where H is the midpoint on the AC side of the triangle, according to the coordinates of the point H and the coordinates of the point D, it is easy to get the range of the yellow rectangle r0 in the figure. All tiles that exist in the region (tiles must be included as a whole) will be removed directly. If the rectangular frame is large, that is, after removing r0, a large number of Tiles still exist in the red rectangular frame AGCD, and a similar operation can be performed again. As shown in Fig.10(b), calculate the AH edge midpoint J, the HC edge midpoint I, the range of the area of the yellow rectangle r1 is obtained, and all the tiles existing in the range are eliminated, and so on. Finally, the Tile that needs to be calculated by the detection algorithm is only marked by the red dot in the figure. If the bounding box generated by the triangle ABC contains more Tile numbers, or the bounding box size is constant, the triangle area is smaller (for example, the narrow triangle), then the treatment effect is more obvious.



a. Large area triangle primitive



a. Smaller area triangle primitive

Fig.10 Improve Representation method of dividing triangle primitive BBOX

In summary, when the number of Tiles to be calculated is large, the improved primitive representation method proposed in this section can effectively reduce the cumulative amount of calculation and improve the processing speed of the TGC. By calculating the calculation amount required for calculating each region r0, r1⋯rn in Fig.10(b), as shown in Table 1, it is used for calculation as the number of Tiles in different regions is eliminated. The amount of calculation in the area will be multiplied, and the elimination operation will be used iteratively. After reaching a certain threshold, the amount of calculation will increase. Therefore, it is particularly important to judge the number of times the culling area is used, and the judgment is based on comparing the calculation amount for calculating the current culling area with the number of times of calculation for calculating the remaining tiling, wherein the calculation amount of the til is calculated as Table 2 shows. For example, if it is determined whether the Nth culling area needs to be calculated, and the number of tiles that are not currently eliminated is M, only an approximate comparison of 4 (N-1) and 4M size is required. If 4(N-1) ≥ 4M, then each tile is calculated directly using the detection algorithm; if 4(N-1) < 4M, then the calculation of the culling region can be performed.

Table 1 Calculation of the amount of calculations each time the culling area is calculated

data item	Addition times	Number of divisions
First tile elimination	1	1
Second tile elimination	2	2
Nth Tile Elimination	2（N-1）	2（N-1）

Table 2 Calculation of the amount of calculations each time the tile is overwritten

Number of tiles to be calculated	Multiplication times	Number of subtractions
one Tile	3	1
two Tile	6	2
M Tile	3M	M

C. *Theoretical Derivation and Feasibility Analysis of Detection Algorithm*

Assuming that the three vertexes of the triangle are on the same plane, select one of the vertexes, and the other two vertexes are only relative displacements to the point. For example, if point A is selected as the starting point, then point B is equivalent to moving a distance in the AB direction, and point C is equivalent to moving a distance in the direction of the AC. Therefore, for any point P in the plane, it can be expressed by Equation 1, where the coefficient U or V is a negative value, which is equivalent to moving in the opposite direction. It is easy to find that when U = V = 0, P = A, when U = 0 and V = 1, P = B, and when U = 1 and V = 0, P = C, if and only if equation 2 is satisfied, the point P is located inside the triangle ABC.

By using equation 3, we can transform equation 1 into equation 4. The two sides of the equation are respectively multiplied by V_0 and V_1. Note that U, V are natural numbers and others are vectors, then formula.5 can be expanded for computation equation.

$$P = A + U*(C-A) + V*(B-A) \quad (1)$$

$$\begin{cases} U \geq 0 \\ V \geq 0 \\ U+V \leq 1 \end{cases} \quad (2)$$

$$P - A = U*(C-A) + V*(B-A)$$
$$\begin{cases} V_0 = C - A \\ V_1 = B - A \\ V_2 = P - A \end{cases} \quad (3)$$

$$V_2 = U*V_0 + V*V_1 \quad (4)$$

$$\begin{aligned} V_2 \cdot V_0 &= U*(V_0 \cdot V_0) + V*(V_1 \cdot V_0) \\ V_2 \cdot V_1 &= U*(V_0 \cdot V_1) + V*(V_1 \cdot V_1) \end{aligned} \quad (5)$$

$$\begin{aligned} U &= \frac{((V_1 \cdot V_1)(V_2 \cdot V_0) - (V_1 \cdot V_0)(V_2 \cdot V_1))}{((V_0 \cdot V_0)(V_1 \cdot V_1) - (V_0 \cdot V_1)(V_1 \cdot V_0))} \\ V &= \frac{((V_0 \cdot V_0)(V_2 \cdot V_1) - (V_0 \cdot V_1)(V_2 \cdot V_0))}{((V_0 \cdot V_0)(V_1 \cdot V_1) - (V_0 \cdot V_1)(V_1 \cdot V_0))} \end{aligned} \quad (6)$$

It can be observed from formula.6 that when three vertexes of a triangle are given, for any point P in the plane, only two of the numerators can be changed to determine the positional relationship between the point P and the triangle. Just change the V2 in the molecule on the right side of the equation to determine the positional relationship between point P and the triangle.

The partitioning method proposed in this paper can pre-empt a part of the tiles that do not overlap with the primitives, and reduce the cumulative amount of coverage calculation. As shown in Fig.11, after calculated three midpoints of D, E, and F, all the tiles contained in the yellow rectangular frame are eliminated. If the above processing is continued, the amount of calculation for the midpoint will increase, and the computational cost of the tiles that need to be eliminated and tested will exceed the cost of direct testing. Therefore, for the tiles represented by orange in the Fig.12, which is necessary to determine whether or not to overlap with the triangle by the coverage calculation.

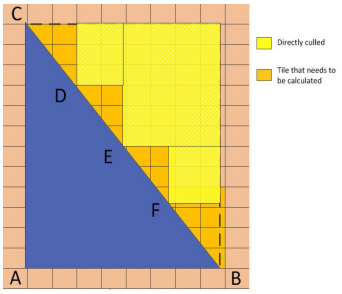


Fig.11 Calculation of remaining tiles

The core problem of the algorithm proposed in this paper is to find the value relationship of U, V and U+V in formula.2. As you can observe from formula.6 that the denominators of the two equations are identical and the size of the value is only related to the coordinates of the three vertexes of the triangle, which brings us three benefits:

(1) From the vector form of Cauchy inequality, we can get the formula.7 to be constant, so the denominator part must be positive. Therefore, we only need to judge the positive or negative of the numerator.

(2) Because U+V<=1, as shown in formula.8, the inequality can be multiplied by the denominator to avoid the division of floating point numbers.

(3) Since the denominator part has nothing to do with the value of point P, the calculation of V0, V1 and the denominator can be completed in advance in the transformation process in the primitive assembly.

$$(V_0 \cdot V_0)(V_1 \cdot V_1) >= (V_0 \cdot V_1)(V_1 \cdot V_0) \quad (7)$$

$$\begin{aligned} &((V_1 \cdot V_1)(V_2 \cdot V_0) - (V_1 \cdot V_0)(V_2 \cdot V_1)) + \\ &((V_0 \cdot V_0)(V_2 \cdot V_1) - (V_0 \cdot V_1)(V_2 \cdot V_0)) - \\ &((V_0 \cdot V_0)(V_1 \cdot V_1) - (V_0 \cdot V_1)(V_1 \cdot V_0)) <= 0 \end{aligned} \quad (8)$$

In summary, the pseudo code of the algorithm is:

```
A=（V1·V1）（V2·V0）-（V1·V0）（V2·V1）
B=（V0·V0）（V2·V1）-（V0·V1）（V2·V0）
C=（V0·V0）（V1·V1）-（V0·V1）（V1·V0）
for row_next_C to row_End _B
if ( ( B && C ) < 0 )
    The tile is a false-overlap tile for the primitive
else
    Calculation D=B+C-A
        if ( D > 0 )
            The tile is a false-overlap tile for the primitive
        else
            The tile overlap with the primitive
```

V. Experimental results and analysis

The verification of the reliability of the detection algorithm is divided into two parts: First, whether the Tile processed by the algorithm can be correctly written into the Tile list, for which we use a rasterizer that can process 16×16 size Tile blocks in the Tile list. The vertexes of the primitives on multiple Tiles are scanned and transformed, and the attribute differences are filled with Matlab. Secondly, the generated Tile list contains the amount of Tile information and the calculated cumulant whether is less than the amount of Tile information obtained by other algorithms. For this purpose, we count the amount of Tiles used to store primitives in the Tile list and display them on the FPGA prototype system.

A. Tile list structure

The Tile list structure constructed by combining multiple tiles of a certain primitive, when rasterizing, can quickly

operate the linked list by indexing the primitives, thereby improving the rendering execution rate, but Store more intermediate data. Therefore, the Tile list built in this article stores the tiles that need to be rendered.

When the screen is divided into tiles, it is inevitable that a triangle primitive covers multiple tiles. The proposed algorithm is used to calculate the coverage relationship between the triangle primitive and the tile, and the triangle coverage factor is used for description. If the triangle element to be drawn intersects with the current tile, and the graphic in the tile is not a triangle, the triangle division circuit first subdivides the primitive on the tile into multiple triangles, and then writes into the Tile list to make the rasterization only triangles are manipulated during processing to improve rendering efficiency. After the Triangle Division, we need to create a tile list to store the current tile number, the element information, the triangle element number subdivided on the Tile, and the position coordinates of the triangle and their attributes.

The processing procedure is as shown in Fig.12, assuming that the currently tested triangle primitive 1 and primitive 2 are located in Tile(m-1,n), Tile(m,1), Tile(m,2) respectively. First, the triangle information of the original primitive 1 and the primitive 2 is stored; then, the triangle is subdivided by the Triangle Division circuit, and the original primitive number, triangle number Tri #1~#n+1， the corresponding coordinate position and its properties corresponding to each Tile after subdivision are stored in the corresponding Tile (m-1, n), Tile (m, 1) and Tile (m, 2), which are generated into a Tile list.During the subdivision process, Tile(m-1,n) contains primitives 1 and 2, so the display relationship needs to be tested during pixel rendering. Tile(m, 2) only contains element 2, and the stored pixels will be displayed on the screen without depth testing. In the Tile list, the tag bit is added, and the tile with the tag bit marked 1 is tested for visibility. The tile with the tag bit marked 0 can be skipped directly.

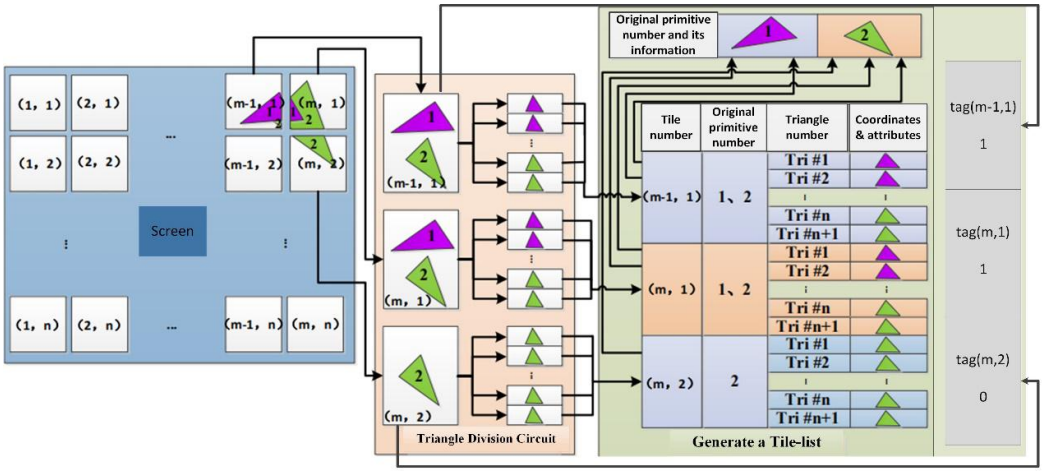


Fig.12 Tile list generation process

B. Result data rasterized rendering output

The implementation steps are as follows: the original vertex coordinates are transformed by vertex data, the points and lines are converted into triangles, and then the Tile-list is generated in the TA platform, but a certain triangle element may span multiple tiles, all related tiles are selected in the Tile-list, then scanned by the rasterizer in the pixel shader, and restored by Matlab software. Finally, the difference between the software result and the theoretical figure will be analyzed and compared, and the conclusion will be drawn. Fig.13 shows the verification process based on this algorithm.

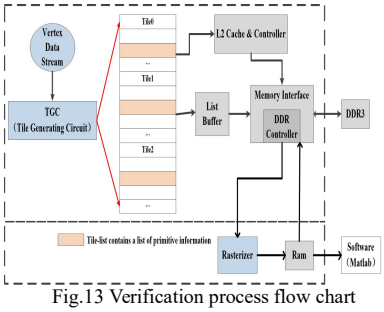
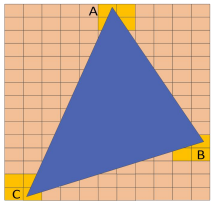
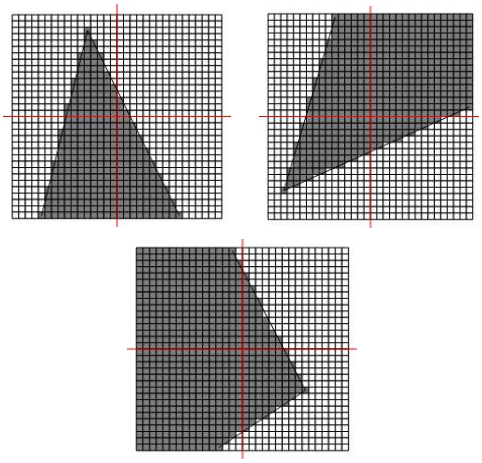


Fig.13 Verification process flow chart

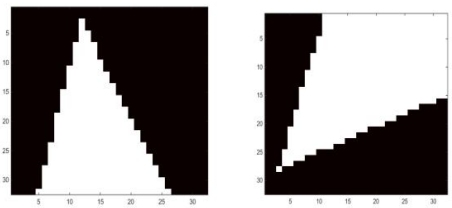
The rendering result output by the rasterization circuit is shown in Fig.14, where Fig.14(a) represents a triangle primitive covering multiple Tiles. Because the amount of Tile data is too large, it is difficult to verify Tile by Tile. The four tiles that make up each vertex are grouped together (yellow part) for verification. Fig.14(b) is the result of three sets of vertexes drawn according to the intercepted waveform diagram after functional simulation. It should be noted that 64×64 small squares do not represent Tile, but four Tiles with size 16×16. Fig. 14(c) shows the result of performing FPGA test by capturing the RGB data stored in Ram and filling it with Matlab. After analysis and comparison, the detection algorithm for large triangle primitives designed in this section can accurately generate the Tile list and record the attribute information on each Tile, and call the linked list information through the rasterizer to correctly complete the scan traversal of the triangle.

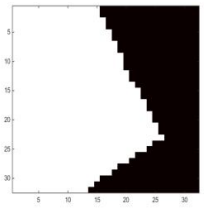


a. Triangle primitive



b. Drawing results for different tiles





c. Matlab restores FPGA test results

Fig.14 Output comparison

C. *FPGA prototype system output*

Fig.15 is a platform for the prototype system of the IP core (mainly including: TGC circuit, rasterizer and depth test).

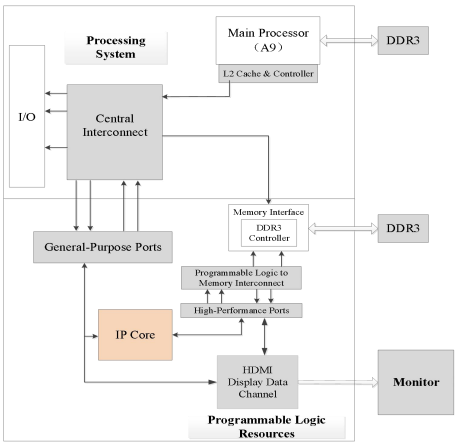
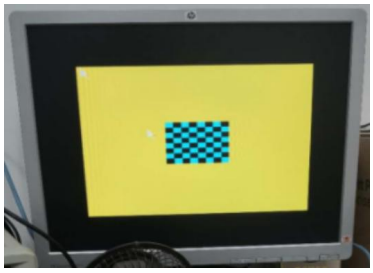


Fig.15 System verification flow chart

The data path is described as follows: First, the vertex stream processed by the vertex shader is stored in the DDR3 of the PS (Processing System) through the ARM processor, and the control information is sent to the IP core through the AXI bus. Second, the processing of the IP core is performed. The process includes: reading the vertex data in the DDR3, generating a Tile list after the element assembly process, writing it back to the DDR3 of the PL (Programmable Logic) side through the memory interface, and reading the PL by the visibility test and the rasterization circuit. The linked list information in the DDR3 is tiled, and the result data is written back to the PS side DDR3 through the AXI-Interconnect; finally, the rendered graphic pixel data is output to the display through the HDMI display path. The platform diagram of the development board prototype system is shown in Fig. 16.

a. Street light



b. Multiple rectangular combined images

Fig.16 FPGA prototype system output image

D. Experimental results analysis

Based on the TA circuit platform (supporting tiles of 32×32 and 16×16 sizes), the triangle primitive representation method and detection algorithm designed in this paper are mapped. Reference [18] for the 32×32 size Tile, for its own detection algorithm, in the case of window sizes of 640×480, 1280×1048 and 1200×1600, test the 5-frame benchmark model Doom3 for each frames, all tiles are rendered independently. The TA platform designed in this paper supports screen scaling rates of 640×480 and 1280×1048. The platform implementation and Tile list generation will carry out on FPGA, and statistics of the average of the primitives in the Tile list generated based on the BBox test and the detection algorithm of this paper. Quantity, and compared with the literature [17]. Table 1 is a comparison of the reliability of the three detection algorithms.

Table.1 Average Amounts of Primitives in All Tile Lists

Benchmark	Doom3	Doom3	school logo
Screen Size	640×480	1024×1048	640×480
BBOX	47514.80	141434.00	4586.57
In Ref. [17]	26769.84	59982.16	-
Algorithm	26734.81	59974.83	2471.54

By comparing the average number of primitives in the Tile list in Table 3, we can see that the average number of primitives stored in the Tile list generated by the detection algorithm in this paper and the algorithm in the Reference [17] is reduced by 40%~60% compared with the traditional BBOX method. When the picture element is larger, the advantage is more obvious. Comparing this paper with the number of primitives produced in [17] is roughly the same, because these algorithms used can calculate relatively accurate coverage. However, the difference is that the algorithm proposed in this paper can pre-eliminate most of the Tiles that does not overlap the primitives, which reduces the amount of computation and improves the rendering efficiency of mobile GPU.

In Ref.[21] proposes a tile binning algorithm that eliminates the minimization of graphics without overlapping tiles, and counts memory occupancy and CPU latency. Table.2 shows the performance comparison of the intermediate data generated by the algorithm in terms of storage space, CPU latency and computing resources.

Table.2 Performance Comparison

-	Memory Overhead	CPU latency	Reduced computing resources
In Ref. [21]	16~35%	23~54%	-
Algorithm	15~35%	24~55%	32~45%

Both the literature [21] and the detection algorithms proposed in this chapter can reduce the memory overhead by nearly 30%, because the traditional BBOX method is used, and in general applications, there will be an error of about 30% error [22]. That is, there are a lot of tiles in the Tile list that don't need to be rendered.

Compared with the literature [21], the calculation of this chapter is reduced by 32%~45%. The main reasons are as shown in Table 3:

Table 3 Whether each Tile and the primitive overlaps

calculation for each tile	Multiplication times	Number of subtractions	Area of elimination
In Ref. [17]	3	1	(y1-y0)(x1-x0)/4
Algorithm	4	1	-

In Table 3, (x0, y0), (x1, y1) represent the coordinates of two points used to calculate the starting vertex of the culling area. The reduced computation is obtained in two parts. The first part is to reduce one multiplication calculation for each calculation of a Tile detection algorithm; the second part is to pre-empt the area occupied by the Tile, which is also a key factor to reduce the calculation amount.

V. CONCLUSION

We propose a new method for segmentation of primitives, on this basis, an overlap detection algorithms is presented. According to the analysis of experimental results on FPGA, this algorithm can reduce memory overhead by 15% to 35%, CPU latency by 24% to 55%, and computing resources by 32% to 45%.

ACKNOWLEDGMENT

The authors would like to acknowledge the grants from the National Natural Science Foundation of China (61602377) , Shaanxi Science & Technology Co-ordination & Innovation(2016KTZDGY02-04-02).

REFERENCES

[1]. Williams R S. What's Next? The end of Moore's law][J]. Computing in Science & Engineering, 2017, 19(2):7-13.

[2]. Owens J . Streaming Architectures and Technology

Trends[M]// GPU Gems \textln2. 2005.
[3]. Yoo J J, Lee J, Krishnadasan S, et al. Tile-based path rendering for mobile device. Proceeding of SA '15 SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications , Article No. 5, November 02 - 06, 2015, Kobe, Japan. New York, NY, USA:ACM, 2015, 6p.
[4]. Woo R, Choi S, Sohn J H, et al. A 210mW graphics LSI implementing full 3D pipeline with 264Mtexels/s texturing for mobile multimedia applications, Digest of Technical Papers of IEEE International Solid-State Circuits Conference 2003, San Francisco, CA, USA, IEEE, 2003:358-367.
[5]. Kim H Y , Yu C H , Kim L S . A memory-efficient unified early z-test.[J]. IEEE Transactions on Visualization & Computer Graphics, 2011, 17(9):1286-94.
[6]. Kim J S , Kim D H , Lee K Y , et al. A hierarchical tiling algorithm for tile based rendering with Global Scratch Counter under multi core environment[C]// Tencon IEEE Region 10 Conference. IEEE, 2012.
[7]. Iosif Antochi. Suitability of tile-based rendering for low-power 2D graphics acceleratoers [D]. Technology University Delft, Delft, the Netherlands, 2007.
[8]. Hsiao C C , Chung C P , Yang H C . A Hierarchical Primitive Lists Structure for Tile-Based Rendering[C]// 2009 International Conference on Computational Science and Engineering. IEEE, 2009.
[9]. Juurlink B , Antochi I , Crisu D , et al. GRAAL: A Framework for Low-Power 3D Graphics Accelerators[J]. IEEE Computer Graphics and Applications, 2008, 28.
[10]. Bratt I . The ARM® Mali-T880 Mobile GPU[C]// Hot Chips 27 Symposium. IEEE, 2016.
[11]. Hsiao C C , Chu S L , Dai S S . Efficient rendering and cache replacement mechanisms for hierarchical tiling in mobile GPUs[C]// Global High Tech Congress on Electronics. IEEE, 2013.
[12]. Hsieh E, Pentkovski V, Piazza T. ZR:a 3D API transparent technology for chunk rendering[C]// Acm/ieee International Symposium on Microarchitecture. 2001.
[13]. Cox M, Bhandari N. Architectural implications of hardware-accelerated bucket rendering on the PC[C]// Acm Siggraph/eurographics Workshop on Graphics Hardware. 1997.
[14]. Chen M, Stoll G, Igehy H, et al. Simple models of the impact of overlap in bucket rendering[C]// Acm Siggraph/eurographics Workshop on Graphics Hardware. 1998.
[15]. Antochi I , Juurlink B , Vassiliadis S , et al. Scene management models and overlap tests for tile-based rendering[C]// Euromicro Symposium on Digital System Design. IEEE, 2004.
[16]. Eberly D H. 3D Game Engine Design, Second Edition: A Practical Approach to Real-Time Computer Graphics (The Morgan Kaufmann Series in Interactive 3D Technology)[M]. 2006.
[17]. Hsieh H C , Hsiao C C , Yang H C , et al. Methods for Precise False-Overlap Detection in Tile-Based Rendering[C]// 2009 International Conference on Computational Science and Engineering. IEEE, 2009.
[18]. Yoo H, Woo J, Sohn J, et al. Mobile 3D Graphics SoC : From Algorithm to Chip. Wiley-IEEE Press, 2010:352.
[19]. Moya V, González C, Roca J, et al. A Single (Unified) Shader GPU Microarchitecture for Embedded Systems. Lecture Notes in Computer Science, 2005, 3793:286-301.
[20]. Pulli K, Aamio T, Miettinen V, et al. Mobile 3D graphics with OpenGL ES and M3G[M]. San Francisco: Morgan Kaufmann, 2007.
[21]. Yoo J J , Lee S , Jung S , et al. Tile binning algorithm for vector graphics minimizing false overlap[C]// IEEE International Conference on Consumer Electronics. IEEE, 2013.
[22]. Gottesfeld S. Simple models of the impact of overlap in bucket rendering[C]// Acm Siggraph/eurographics Workshop on Graphics Hardware. ACM, 1998.