IEICE TRANS. INF. & SYST., VOL.E101–D, NO.1 JANUARY 2018

LETTER

A GPU-Based Rasterization Algorithm for Boolean Operations on Polygons

Yi GAO[†], ***Nonmember***, **Jianxin LUO**[†a)], ***Member***, **Hangping QIU**[†], **Bin TANG**[†], **Bo WU**[†], ***and*** **Weiwei DUAN**[†], ***Nonmembers***

SUMMARY This paper presents a new GPU-based rasterization algorithm for Boolean operations that handles arbitary closed polygons. We construct an efficient data structure for interoperation of CPU and GPU and propose a fast GPU-based contour extraction method to ensure the performance of our algorithm. We then design a novel traversing strategy to achieve an error-free calculation of intersection point for correct Boolean operations. We finally give a detail evaluation and the results show that our algorithm has a higher performance than exsiting algorithms on processing polygons with large amount of vertices.

key words: *GPU, CPU, rasterization, Boolean operation, error-free*

1. Introduction

The Boolean operations on planar polygons are frequently used in spatial analysis. According to the data structures, the algorithms can be divided into vector data structure algorithm and raster data structure algorithm. The former uses the geometric analysis to focus on the location relationship among the polygons [1], [2]. Peng et al. [3] proposed a classification method of edges based on the rotation angle. Murta [4] realized an improvement of Vatti [2] to handle coincident edges. Such algorithms have the advantage of high precision. However, they demonstrate certain insufficiencies in the aspects of the complexity of the data structure and the amount of computation.

The raster data structure can simplify the spatial calculation of the polygons more effectively. Fan et al. [5] and Wang et al. [6] transformed the Boolean operations on polygons into discrete pixel operations and effectively improved the performance of the algorithm. However, these algorithms based on CPU have three main problems: firstly, the discussion of complex point-inclusion problem is still inevitable while traversing the discrete pixels; secondly, the time spent of rasterization based on CPU is too much due to the serializability of CPU [7]; thirdly, the process of rasterization is a transformation process with loss that causes a certain degree of error [8].

To addressing these problems, this paper presents a novel algorithm based on GPU rasterization. The algorithm avoids the complex geometric relationships by using the pixel mapping relationships. More importantly, the large-scale parallel features of GPU can effectively achieve high-performance computing. Currently, no relevant research is present that uses this method for Boolean operations on the polygon. To be specific, the proposed algorithm has the following innovative contributions:

- Constructing an interoperation data structure of CPU and GPU which enables a parallel operation in GPU and greatly improves the algorithm efficiency.
- Proposing a simple and fast GPU-based contours extraction algorithm.
- Presenting a traversing strategy which effectively solves even the problem of intersection points on the edge and edge overlapping.
- Providing an errorless computation of intersection point based on this data structure.

2. Data Structure

We design two data structures which are associated through relationship mapping (Fig. 1).

Raster data structure in GPU Three textures were used: Tex1 stores the vertex coordinates with 32-bit floats in R and G channels of each texture fragment. Tex2 stores the previous vertex coordinates in R and G channels and the next vertex coordinates in B and A channels. Tex3 stores the vertex attribute with 8-bit floats in R channel. The first 6-bit describes the fragment value f. The other 2-bit Boolean data tag_1 and tag_2, show whether the vertex belongs to outer or inner contour, using 0 and 1 denote false and true, respectively. A triad (f, tag_1, tag_2) represents the fragment at-

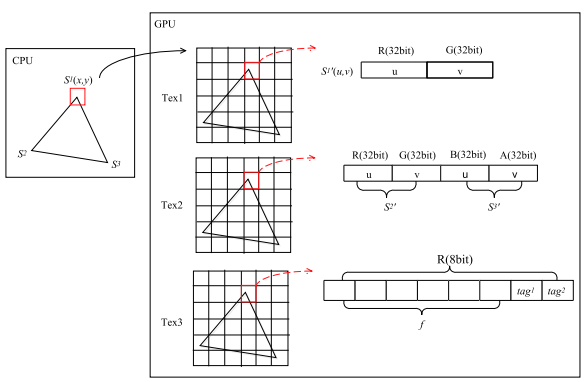

Fig. 1 Mapping relationship between two data structures

Manuscript received May 29, 2017.
Manuscript revised August 29, 2017.
Manuscript publicized September 29, 2017.

[†]The authors are with the PLA University of Science and Technology, Nanjing, P.R. China.
a) E-mail: luojianxin555@163.com

DOI: 10.1587/transinf.2017EDL8119

tribute.

Vector data structure in CPU There are various papers about vector data structures. [2] use *boolean* data to indicate the intersection point of entry or exit property. [6] use *other* pointer to indicate the intersection of another linked list. This paper combines these two structures, defined as follows:

$$\begin{array}{l} vertex = \{x, y : coordinates; \\ tag_1, tag_2 : boolean; \\ pre, next, other : pointer; \} \end{array} ,$$

where x and y denote horizontal and vertical coordinates, respectively; tag_1 and tag_2 are the same as in GPU; *pre* and *next* pointers point to the previous and the next vertex; *other* pointer points to the intersection of another linked list.

There is a mapping relationship: $S' = (MVP)S$, where S is the object space coordinate in CPU, S' is the screen space coordinate in GPU, MV is the transformation matrix, and P is the projection matrix.

3. Extraction of Inner and Outer Contours

Contour extraction includes two steps: polygon rasterization and contour tracking.

Polygon Rasterization A polygon can be easily rasterized into textures in an orthogonal projection camera in OpenGL or DirectX, which is operated in hardware and executed in parallel. The initial raster value f =0. After rasterization, the geometry primitive is assigned correctly. As shown in Fig. 2, the raster value covered by the first polygon S was assigned to 1 (grey rectangles) and covered by the second polygon C was assigned to 2 (pink rectangles). The raster value located in two polygons' intersection region was assigned to 3 (green rectangles). Then, the screen coordinates T_1-T_7 are interpolated along the segment $\overline{S_2S_3}$ (black arrow). Repeating this interpolation process, the geometry primitive is transformed into raster textures.

Given two arbitrary polygons in Euclidean space $S\{S_1, S_2, S_3, S_4, S_5, S_6\}$ and $C\{C_1, C_2, C_3, C_4\}$, all the vertices of two polygons are arranged as counter-clockwise. Let (x, y) be the coordinates of any point in space. Then the Boolean operation of S and C is defined as:

$$\begin{array}{l} S \cup C = \{(x, y, f)|f_{(x,y)} = !0\} \\ S \cap C = \{(x, y, f)|f_{(x,y)} = 3\} \\ S - C = \{(x, y, f)|f_{(x,y)} = 1\} \end{array} \quad (1)$$

Contour Tracking Read all fragments that have f =!0, and compare each fragment with the f value of the surrounding 8 neighbour fragments. As long as there is a fragment with f=0, the current fragment is the outer boundary fragment. These outer boundary fragments constitute a closed loop that is the outer contour (Fig. 3 (a)).

Read all fragments that have f=3, and compare each one with the f value of the surrounding 8 neighbour fragments. As long as there is a fragment with $f \neq 3$, the current fragment is the inner boundary fragment. These inner boundary fragments form a closed loop that is the inner contour (Fig. 3 (b)).

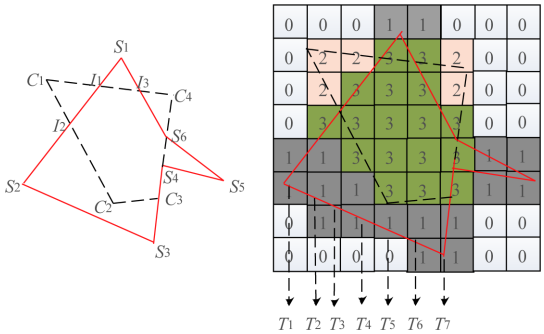

Fig. 2 Two polygons rasterized

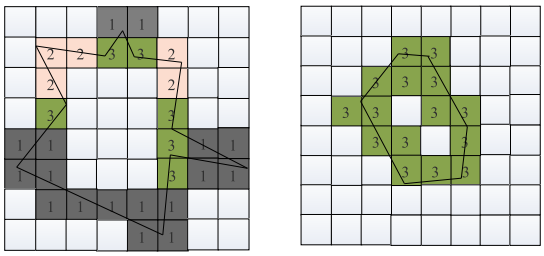

Fig. 3 Extraction of: (a) external contour, and (b) internal contour

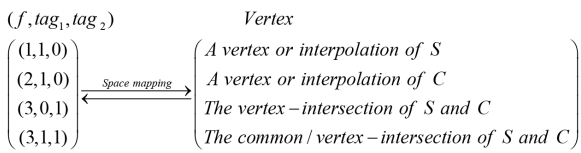

Fig. 4 The corresponding relations between triad and vertex

In GPU parallel architecture, the traversal of 8 neighbourhood for each fragment is independent to each other and the total computation time is the time of only one of them. Whereas, in the CPU serial mode, the traversal times of 8 neighbourhood are accumulated [5].

4. Traversing

Each fragment associates with a triad (f, tag_1, tag_2). The corresponding relations between triad and vertex are shown in Fig. 4. The case of (3,1,0) is unable to appear because as long as f=3, tag_2 must be 1. We divide the intersections into the vertex-intersection and the common-intersection property, distinguishing from the entry or exit property [2], [9].

4.1 Contour Compressed

The extracted fragments of inner and outer contours are indexed. As a result of GPU rasterization, there are many redundant data in two contours. The method of distinguishing is to space mapping one vertex $S_i(x, y)$ in object space to $S_i'(x, y)$ in screen space and calculate the distance of T_s. If $dist(S_i', T_s)$ is less than the length of the raster unit l, the

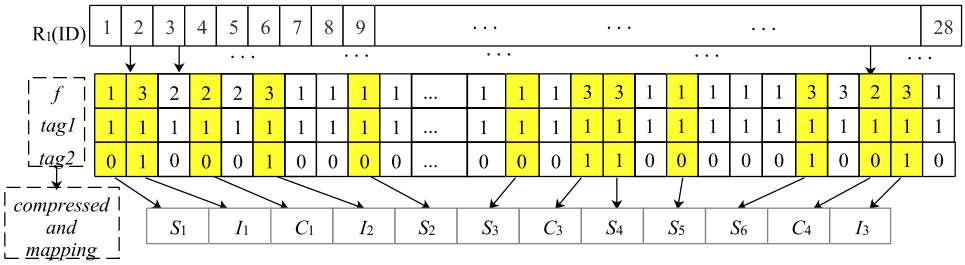

Fig. 5 Illustrations of outer contour traversing and compressed

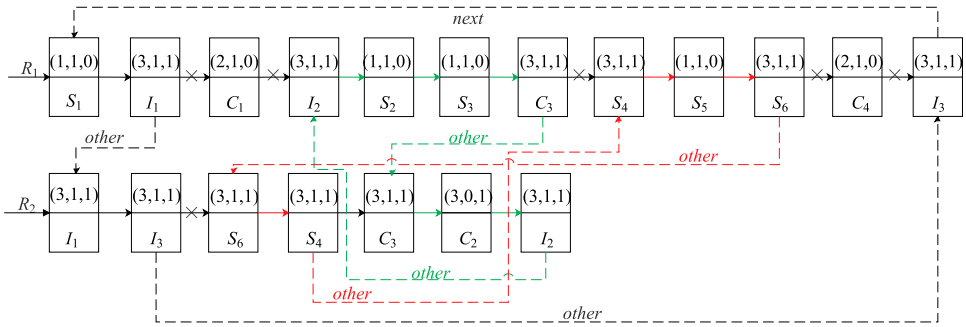

Fig. 8 Illustrations of *S*-*C* difference operation

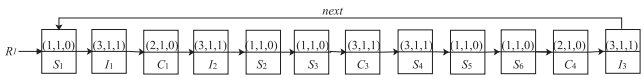

Fig. 6 Illustrations of $S \cup C$ union operation

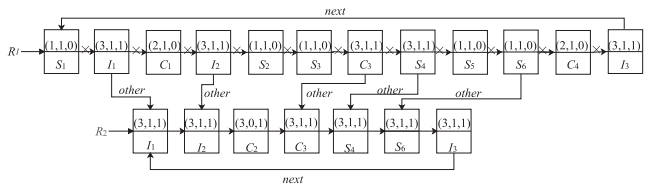

Fig. 7 Illustrations of $S \cap C$ intersection operation

current fragment corresponds to the vertex S_i. Otherwise, it is an interpolation. The fragment that corresponds to the vertex or the intersection was reserved (Fig. 5 yellow block), while the rest as redundant data were deleted, thus the contour was compressed. This compression method can distinguish between the vertex-intersection and an interpolation very well for the problem of intersection points on the edge and edge overlapping.

4.2 Traversing Rules

Let assume that R_1 and R_2 are the copmressed outer and inner contour list of S and C. Note that the default traversing direction is counter-clockwise direction, unless it is specified otherwise.

4.2.1 Union

According to the formula(1), the union of two polygons is the compressed outer contour R_1. We set S_1 as the starting point and get the union of S and C is $\{S_1, I_1, C_1, I_2, S_2, S_3, C_3, S_4, S_5, S_6, C_4, I_3\}$.

4.2.2 Intersection

According to the formula(1), the intersection only lies on f=3. Therefore, the traversing rule is to disconnect two contours for the left and right of f! = 3. We firstly choose I_1 as the starting point, and hit the breakpoint then we shift to R_2'. Repeat these steps until we get a circle (Fig. 7). The intersection is $\{I_1, I_2, C_2, C_3, S_4, S_6, I_3\}$.

4.2.3 Difference

The difference traversing rule is to disconnect two contours for the left and right of $f = 2$ and shift to other chain when a (3,1,1) is hit. As presented in Fig. 8. We firstly choose S_1 as the starting point and begin to traverse I_1 (3,1,1), then we shift to R_2 and find the next node of I_1 is I_3. In this way, we get a circle (the black dashed lines). Afterwards, we take the break I_2 as a new starting point, and then continue to traverse, getting another circle (the green dashed lines). Next, we use the same method to traverse and get circle (the red dashed lines). Thus, the difference between R_1 and R_2 consists of three parts, $\{S_1, I_1, I_3\}$, $\{I_2, S_2, S_3, C_3, C_2\}$ and $\{S_4, S_5, S_6\}$.

5. Exact Intersection Calculation

Generally, the exsiting algorithms reduce the raster error by increasing the resolution. The greater the resolution is, the smaller raster error.

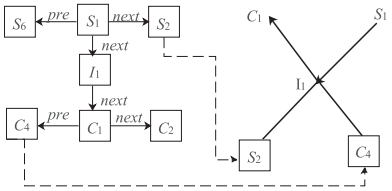

Fig. 9 Intersection calculation

Table 1 Running time results of three main stages (*ms*)

N	rasterization	extracting	traversing
9	0.0003	0.0016	0.0021
35	0.0003	0.0024	0.0035
55	0.0003	0.0027	0.0043
750	0.0126	0.097	0.1589
5510	0.0862	0.1786	0.1957
20,563	0.1431	0.2345	0.3133

According to Fig. 5, we can obtain a resultant chain. Take the common-intersection I_1 example, as shown in Fig. 9, the calculation of I_1 is to find the relevant edges. In our data structure, S_1 and C_1 have their own entity polygon chain. According to the segment direction formed by I_1, S_1 and its *next* vertex S_2 is $\overline{S_1S_2}$, C_4 and its *next* vertex C_1 is $\overline{C_4C_1}$. The relevant edges of I_1 are $\overline{S_1S_2}$ and $\overline{C_4C_1}$. The value of I_1 is obtained using the corresponding linear equation of $\overline{S_1S_2}$ and $\overline{C_4C_1}$. And so on, for each intersections.

6. Experimental Results and Discussion

We have implemented our algorithms in C++ and GLSL. All tests were performed on a desktop computer equipped with Intel Core i5 CPU, 4 GB memory, and NVIDIA GeForce GT 620M GPU.

The raster resolution was set to 256×256, and the average running time of the proposed algorithm was tested in three main stages: rasterization, extracting, and traversing (see Table 1). N denotes the sum of two polygons' vertex numbers.

The results for each stage verify our algorithm is feasible and effective to some extent. Table 2 lists the comparison of extraction performance and intersection area error with Fan [5] and our method, which are both rasterization algorithm. The area error is evaluated based on the area value of Vatti [2], whose value is 7.5m^2. From this table, we can see that our method has better performance. It is ascribed to the interoperation data structure which enables a parallel operation in GPU. The poorer performance of Fan is due to that the extracting process in CPU serial way is inefficiency. Meanwhile, our algorithm has no accuracy error, while Fan even has 0.02% error at resolution 1024×1024. This shows that our algorithm is not affected by resolution, and can achieve an error-free calculation of intersection points.

To further verify the overall performance of our algorithm, we hence conduct another experiment. Table 3 lists the results of our algorithm in comparison with CPU-based vector algorithm Vatti [2] and Peng [3], and CPU-based rasterization algorithm Fan [5]. As we expected, the results also show that the new algorithm outperforms other three ones, and further strengthens our claim that our algorithm can improves the algorithm efficiency.

Table 2 Contour extraction performance results and area error statistics

	256×256		512×512		1024×1024	
	Fan	our	Fan	our	Fan	our
time(s)	15.145	2.132	233.67	7.036	4053.33	22.737
area(m^2)	7.525	7.5	7.5083	7.5	7.5015	7.5
Error%	0.33	/	0.11	/	0.02	/

Table 3 Comparison results with several existing algorithms (*ms*)

N	Vatti	Peng	Fan	Our method
9	0.0635	0.0308	0.0062	0.0041
30	0.4336	0.2568	0.0152	0.0062
50	0.8325	0.513	0.0304	0.0073
742	1.3783	1.0745	0.3677	0.2671
2516	1.767	1.5547	1.1323	0.3433
20,363	2.606	2.2791	2.0032	0.6891

7. Conclusion, Limitations and Future Work

A GPU-based rasterization algorithm for Boolean operation on polygons is presented. Moreover, two data structures for GPU and CPU were designed. According to the vertex mapping and fragment attribute, the proposed algorithm realizes contour fragment compression, and enables correct operations of union, intersection and difference. According to the vertex order in Boolean computation results, the relevant edge was found, and the intersection point values were accurately calculated.

Due to the own limitations of rasterization itself, a potential drawback is that the edge distribution of a polygon in extreme case, such as too many edges within a grid, the efficiency advantage of the proposed algorithm is not obvious.

The algorithm is good at solving two polygons. For multi-polygons, repeat and compute this algorithm in many times will affect the efficiency. In the future, we will research on the algorithm for processing multi-polygons simultaneously.

Acknowledgements

This research work was supported by Chinese National Defense Foundation of Science and Technology (Grant No. 3602027) and Jiangsu Province Science Foundation for Youths (Grant No. BK20150722).

References

[1] K. Weiler and P. Atherton, "Hidden surface removal using polygon area sorting," ACM SIGGRAPH Computer Graphics, vol.11, no.2, pp.214–222, ACM, 1977.

[2] B.R. Vatti, "A generic solution to polygon clipping," Commun. ACM, vol.35, no.7, pp.56–63, 1992.

[3] Y. Peng, J. Yong, H. Zhang, and J. Sun, "Efficient algorithm for general polygon clipping," Proc. 6th International Conference on Computer-Aided Industrial Design and Conceptual Design 2005,

2005.

[4] A. Murta, "A general polygon clipping library," Advanced Interfaces Group, Department of Computer Science, University of Manchester, Manchester, UK, 2000.

[5] J. Fan, W. Kong, et al., "Rapc: A rasterization-based polygon clipping algoirithm and its error analysis," Acta Geodaetica et Cartographica Sinica, vol.44, no.3, pp.338–345, 2015.

[6] R. Wang, X. Liao, et al., "Polygon clipping algorithm based on dual-strategied tracing and grid partition," Journal of Graphics, vol.33, no.6, pp.45–50, 2012.

[7] C. Zhou, Z. Chen, Y. Pian, N. Xiao, and M. Li, "A parallel scheme for large-scale polygon rasterization on cuda-enabled GPUs," Transactions in GIS, vol.21, no.3, pp.608–631, 2016.

[8] S. Liao, Z. Bai, and Y. Bai, "Errors prediction for vector-to-raster conversion based on map load and cell size," Chin. Geogr. Sci., vol.22, no.6, pp.695–704, 2012.

[9] Z.-J. Wang, X. Lin, M.E. Fang, B. Yao, Y. Peng, H. Guan, and M. Guo, "Re2l: An efficient output-sensitive algorithm for computing boolean operations on circular-arc polygons and its applications," Comput. Aided. Design., vol.83, pp.1–14, 2017.