GPU-based Arbitrary Polygon Intersection Area Algorithm

Yi Gao¹,²,a,*, Bo Wu¹,b, Jianxin Luo¹,c and Hangping Qiu¹,d

¹PLA Army Engineering University, Nanjing, China
²PLA troops of 61175, Nanjing, China

aiamninigao@163.com, bmr_wubo@163.com, cluojianxin@163.com,
dQHP8887@sina.com

Keywords: polygon, intersection area, rasterization, Monte Carlo

Abstract. Computation of the intersection area of polygons is an important mathematical problem. For a long time, the processing of arbitrary polygons has been an important and difficult research topic. In this paper, we proposed a GPU-based rasterized polygon intersection area algorithm, GPURAS, and its accelerated versions, GPURASMC, which use Monte Carlo method and prove the correctness of these algorithms. Experiments and comparisons were performed using simple, arbitrary complex, and large-scale polygons. The results show that, compared to CPU-based algorithms, the efficiency of our algorithms is hundreds of times superior.

1. Introduction

The computation of the intersection area of planar polygons is used in a wide range of applications. In the fields of virtual reality, computing the intersection area of multiple polygons is an essential and fundamental technique. Such a technique contains two parts: the intersection test and the intersection area’s computation. The intersection test gives a Boolean output value to determine whether the polygons under consideration intersect, whereas the intersection area computation determines the shape and size of the intersection.

Presently, there are various methods to handle the problem of polygon intersection, but most of them are specialized for convex polygons, such as Shamos[1] and O’Rourke[2] algorithms. For concave polygons, most methods use triangularization or convexification subdivision approaches. However, subdivision approaches are usually complicated and can greatly increase the amount of edge intersection tests, which increases the complexity of the algorithm, especially when a polygon contains many concave points or when two polygons share many intersection points. Therefore, scholars have developed a series of accelerating algorithms[3][4][5]. For example, Murta[7] is the improvement of Vatti[6] algorithm, solving the problem that the horizontal edges are not allowed to enter the polygon and enhancing the stability of the algorithm. Qi et al. [8] propose a new improved algorithm, using the single linked list to avoid the complex point of entry and exit. Cui[9] and Fan et al. [10] propose a new rasterization processing, which is simple and effective to simplify the calculation of the polygon.

However, the above mentioned CPU-based serial processing methods cannot meet the requirements of rapid real-time applications. With the recent developments in the field of computer animation, virtual reality, etc., which aims to show more and more vivid details, the scale of polygon data has been increasing tremendously, often reaching the size of a few or even hundreds of GBs per dataset.

With the emerging development of graphic hardware, the strong computational capability of GPUs has attracted much attention from researchers. Much progress has been made in the field of image processing, algebraic operations, data mining, etc., using GPU based techniques[11][12]. Unlike the CPU’s serial processing model[13], the parallel processing model of the GPU can significantly reduce the execution time and greatly improve program efficiency.
Hence, the study of a GPU-based arbitrary polygon intersection area computation algorithm has not only theoretical importance in terms of computational complexity, but also practical significance in many engineering and/or design applications in the fields of computer vision, computer graphics, image processing, etc.

2. **Polygon Intersection area algorithm based on CPU**

The existing approaches of calculating a polygon’s area can be classified into two categories, deterministic algorithms and numerical algorithms.

The idea of deterministic algorithm is that determines whether each side of polygon A intersects with polygon B, if true, then judges the relationship between line segments (completely overlap, one endpoint overlap and collinear, parts overlap, intersect with two endpoints, intersection with line, orthogonal or no intersection) and then judge whether the vertex of the A (B) on the inside of B (A). According to the intersection field, obtaining the coordinate of each vertex of this new polygon \( P_1P_2P_3\ldots P_n \). Vertices are ordered counter-clockwise. The polygon area can be calculated by dividing the polygon into several adjacent triangles and computing the algebraic sum of those triangles’ areas. The area of a triangle can be obtained by the outer product of the two plane vectors composed of the three vertices.

Numerical algorithms usually use the Monte Carlo to calculate a definite integral (i.e., the algebraic sum of the integrand within the interval), which is based on the stability suggested by the law of large numbers in probability theory. Considering a uniform random variable \( \xi \) in the plane area \( \Omega=[a,b]\times[0,T] \), then \( p = P(\xi \in S) = \frac{1}{(b-a)T} \int_a^b f(x)dx \). We can then generate two random numbers \( x_i, y_i \). Let \( x_i, y_i \) be an available coordinate of the random point \((\xi, \eta)\) and check whether the random point \((x_i, y_i)\) not only falls into A but also falls into B.

Due to the complexity of the heavy computational load of the deterministic algorithm, it is difficult to apply it to the computation of large-scale polygons’ intersection areas. The numerical algorithm can obtain accurate and reliable results with sufficient simulation times. However, it needs to conduct point inclusion tests for a large number of random points, so the overall performance is limited by the number of polygons.

3. **GPURAS**

In this section, we will discuss an efficient GPU-based algorithm for computing the intersection area of arbitrary polygons(GUPRAS) in detail. For clarity and convenience, the meaning of the frequently used concepts are given below:

- \( P, Q \): input polygons with vertices sorted in counter-clockwise order
- \( R \): a point
- \( V \): vertex of a polygon
- \( N \): numbers of vertices or edges in a polygon
- \( V_x, V_y \): the x-coordinate and y-coordinate of a polygon vertex
- \( \text{RasterWidth} \): the width of the raster sheet, defined by the minimum and maximum coordinates in the x-direction among all the vertices, i.e. \([V_{x_{\min}}, V_{x_{\max}}]\)
- \( \text{RasterHeight} \): the height of the raster, defined by the minimum and maximum coordinates in the y-direction among all the vertices, i.e. \([V_{y_{\min}}, V_{y_{\max}}]\)
- \( S_{\text{TexArea}} \): area of the raster sheet, determined by the sheet’s width and height
- \( S \): intersection area of polygons
- \( F \): raster location identifier
- \( \{P\} \): raster set of polygon P
- \( \{Q\} \): raster set of polygon Q
I (P): interior region of P
E (P): exterior region of P
I (Q): interior region of Q
E (Q): exterior region of Q

Theorem 1 (point inclusion test): If the corresponding raster identifier $F$ of an arbitrary point $R$ is 0, and its 8-neighboring rasters $F$ are all 0, then this point must be in the exterior region of polygon $P$. On the other hand, if the corresponding raster identifier $F$ of an arbitrary point $R$ is equal to or greater than 1, and its 8-neighboring raster $F$ are all equal to or greater than 1, then this point must be in the interior region of polygon $P$.

To express Theorem 1 in mathematical form: For an arbitrary $R$, if $\sum_{i=0}^{7} F_i = 0$ ⇔ $R \in E(P)$; If $F_R \geq 1$ and $F_i \geq 1, (0 \leq i \leq 7)$ ⇔ $R \in I(P)$, where $i$ is one of the 8 neighbors of $R$.

As shown in Figure 1 (left), $F_R=0, \sum_{i=0}^{7} F_i = 0$, then $R \cap \{P\} = \emptyset$, so $R \in E(P)$. As shown in Figure 1 (right), $F_R=1, \sum_{i=0}^{7} F_i = 1$, then $R \cup \{P\} = \{P\}$, so $R \in I(P)$.

Figure 1. Relative locations of a point with respect to a polygon.

Theorem 1 can be used to rapidly exclude the points outside the polygon to minimize the point set to be further examined, which is more efficient and faster than performing point-inclusion tests on the whole set. The time complexity of the ray method and winding number algorithm are $O(m*N)$ and the complexity of ray scanning method is $O(m*N \log N)$, where $m$ is the size of point set. If the number $m$ can be greatly reduced, the overall efficiency can be largely improved for applications requiring massive amounts of point inclusion tests.

The intersection of polygons $P$ and $Q$ can be determined by location identifiers. In the initial setting, the raster location identifier $F=0$. When $P$ is rasterized, I($P$)'s location identifier is $F=1$, and E($P$)'s identifier $F=0$. If $P$ intersects with $Q$, the corresponding location identifier $F$ of the intersecting part will be incremented by 1 and then $F=2$.

Theorem 2 (intersection test): If $P$ intersects with $Q$, then the corresponding identifier of the intersecting location must be equal to 2, and vice versa. Expressed in mathematical form: $\exists P \cap Q ⇔ F_{I(P) \cap \{Q\}} = 2$.

Proof: As shown in Figure 2, when $P$ is rasterized, I($P$)'s location identifier is $F=1$ (yellow area), and E($P$)'s identifier $F=0$. When $Q$ is rasterized, $F_{I(P) \cap \{Q\}} = 1$ (green area), the corresponding location identifier $F$ of the intersecting part is 2, $F_{I(P) \cap \{Q\}} = 2$ (red area).

Figure 2. Location relation of polygon.

This can be also considered under four singular conditions. As shown in Figure 3, several relative locations of polygons are given, and the red area indicates the intersecting parts.
When P and Q have a common point, \( \exists \{V_i\} \in I(P) \), \( 0 \leq i \leq N \), then \( F_{\{p_i\}|\{q_i\}} = 2 \);

When P and Q have a common line, \( \exists \{L_i\} \in I(P) \), \( 0 \leq i \leq N \), then \( F_{\{p_i\}|\{q_i\}} = 2 \);

When P and Q have a common area, \( \exists \{Q\} = \{P\} \), then \( F_{\{p_i\}|\{q_i\}} = 2 \);

When Q includes P (or is included by P), \( \{P\} \subseteq \{Q\} \) (or \( \{Q\} \subseteq \{P\} \)), then \( F_{\{p_i\}|\{q_i\}} = 2 \).

Compared to algebraic methods, the principle of this intersection area calculation is simpler. It could be obtained according to pixel ratio and raster field’s area. The equation of the intersection area’s calculation is:

\[
S = \frac{\text{count}}{\text{RES}} \cdot S_{\text{TexArea}}
\]  

where count is the raster number of \( F=2 \), and RES is the resolution of raster.

This algorithm converts a polygon of vertices into a rasterized image, generates the sampling rasters in the GPU, determines if the targets intersect with each other according to the rasters’ location, and calculates the intersection area by counting the number of intersected rasters. It does not make any assumptions about convexity or concavity of polygons and can be applied to any rasterizable geometric elements, which are not limited by the concavity and convexity of polygons.

4. GPURSAMC

GPURAS algorithm has better universality but the efficiency is affected by CPU and GPU communication delay. In order to achieve further acceleration, a small number of random raster sub-blocks are used to simulate the whole raster field to reduce the magnitude of the read buffer and improve the efficiency of the algorithm. We randomly select \( m \) points from the raster field and mark the number of raster grids whose identifiers are equal to or greater than 2 as \( \text{count}' \). Then, the probability of the selected point falling in the intersection region is \( S/S_{\text{TexArea}} \). The intersection area can be computed as:

\[
S = \frac{\text{count}'}{m} \cdot S_{\text{TexArea}}, \quad (m \leq \text{RES})
\]  

Proof of correctness: Without loss of generality, let \( S_{\text{TexArea}} \) be the area of the entire raster field and \( S \) be the area of the intersection region of the two polygons.

Event A: Cast one point and it falls in the intersection region.

The location of the casted point follows a two-dimensional uniform distribution, so \( p(A) = S / S_{\text{TexArea}} \). Let \( \text{count}' \) be the number of casted points falling in the intersection area, i.e. points with raster location identifier equal to or greater than 2, with a total of \( m \) points, and \( \epsilon > 0 \) be
an arbitrary positive number. According to the Bernoulli law of large numbers
\[
\lim_{x \to \infty} p \left\{ \frac{\text{count}'}{m} - p(A) < \varepsilon \right\} = 1,
\]
and when \( m \) approaches infinity, the frequency, \( \frac{\text{count}'}{m} \), will converge to the probability, \( S / S_{\text{TexArea}} \). Thus, the intersection area can be computed as
\[
S = \frac{\text{count}'}{m} \cdot S_{\text{TexArea}}, \quad (m \leq \text{RES}), \quad \text{i.e. Equation (2)}.
\]
When \( \text{count}' \) and \( m \) approach infinity, the equation becomes Equation (1).

Complexity: It can be seen from Equation (1) that the complexity of the GPURAS and GPURASMC algorithms is only related to the rasters’ resolution. The complexity of the GPURAS algorithm is \( o(\text{RES}) \). For example, when the raster’s resolution is 2048*2048, then the complexity is \( o(2048 \cdot 2048) \). If the sampling chooses only one percent of the points, it can be known from Equation (2) that the complexity is \( o(204 \cdot 204) \), which means the complexity of GPURASMC algorithm is \( o(m) \).

5. Experiments and discussion

The algorithms were implemented using C++ and GLSL and executed in the environment of Microsoft Windows 7 and OpenGL 4.4.0 with an Inter® Core(TM)i5-3337U processor, 4G memory, and a NVIDIA GeForce GT 620M graphics card. Experiment uses three different polygon models:

5.1 Error rate analysis

The raster processing itself is a lossy transformation and errors are inevitable[14][15]. It is necessary to make the error analysis using our algorithms with the CPU-based deterministic algorithm. The relative area error of \( E \) is defined as follows:

\[
E = \left| \frac{S - S_{\text{CPU}}}{S_{\text{CPU}}} \right| \times 100\%.
\]

\( S \) is the intersection area of our algorithm, \( S_{\text{CPU}} \) is the intersection area of CPU-based deterministic algorithm. Experiment uses two different polygon models:

- Model 1 consists of two randomly-generated simple polygons. The randomly generated polygons are \( P1 = \{(4, 4), (11, 3), (12, 6), (9, 8)\} \) and \( P2 = \{(11, 3), (16, 3), (18, 8), (14, 12), (9, 8)\} \).

- Model 2 consists of two complex polygons with “holes”. In the division of polygon exterior and interior regions, the planar topological relation should be taken care of. If the edge connection direction of the outer boundary is counter-clockwise, then the edge connection direction of “holes” should be clockwise.

Experimental results are shown in Table 1 and Table 2.

<table>
<thead>
<tr>
<th>RES</th>
<th>256*256</th>
<th>512*512</th>
<th>1024*1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(%)</td>
<td>0.45</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>GPURAS</td>
<td>5.5250</td>
<td>5.4936</td>
<td>5.5029</td>
</tr>
<tr>
<td>E(%)</td>
<td>0.08</td>
<td>0.0704</td>
<td>0.0563</td>
</tr>
<tr>
<td>GPURASMC</td>
<td>5.6560</td>
<td>5.5044</td>
<td>5.4963</td>
</tr>
<tr>
<td>E(%)</td>
<td>0.284</td>
<td>0.08</td>
<td>0.067</td>
</tr>
</tbody>
</table>
Table 2. Experimental results of Model2 and statistical results of area error.

<table>
<thead>
<tr>
<th>RES</th>
<th>256*256</th>
<th>512*512</th>
<th>1024*1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>E%</td>
<td>1.7778</td>
<td>1.7847</td>
<td>1.7964</td>
</tr>
<tr>
<td>E/(%)</td>
<td>1.52</td>
<td>1.15</td>
<td>0.49</td>
</tr>
<tr>
<td>GPURASMC</td>
<td>1.7810</td>
<td>1.8109</td>
<td>1.8086</td>
</tr>
<tr>
<td>E/(%)</td>
<td>1.35</td>
<td>0.3</td>
<td>0.17</td>
</tr>
</tbody>
</table>

For two simple polygons, the intersection area of Model1 is 5.5 in CPU-based deterministic algorithm. The error rate of GPURAS is 0.45% when a resolution of 256*256, while the error rate of GPURASMC using 60% of all points reaches 2.84%. Increasing the resolution, the error rate becomes smaller and smaller. For a resolution of 1024*1024, the error rate of GPURAS is merely 0.05%, and the error rate of GPURASMC using 60% of all points is reduced to 0.067%. For two complex polygons with “holes”, our algorithms also provide precise and high-efficient computation. Such an error rate can be considered indicative of round-off errors and is acceptable in most engineering and large-scale software projects.

5.2 Execution efficiency comparison

In this paper, our algorithms is the numerical solution which is the same as CPU-based Monte Carlo algorithm, so we will compare the efficiency of these algorithms under the same hardware experimental conditions. Model 3 gives two convex polygons with many vertices, P1 having 800 vertices and P2 having 358 vertices. For CPUMC and GPURASMC, the experiments were repeated ten times with different randomized samplings to obtain the error estimation. Experimental results are shown in Table 3.

Table 3. Experimental results of Model3 and execution time(s).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$10^1/256*256$</th>
<th>$10^2/512*512$</th>
<th>$10^3/1024*1024$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUMC</td>
<td>6.856</td>
<td>65.151</td>
<td>/</td>
</tr>
<tr>
<td>GPURAS</td>
<td>0.017</td>
<td>0.03</td>
<td>0.035</td>
</tr>
<tr>
<td>GPURASMC</td>
<td>0.011</td>
<td>0.027</td>
<td>0.032</td>
</tr>
</tbody>
</table>

It can be seen from Table 3 that under the same magnitude of resolution (256*256~$10^4$), the execution time of CPUMC is 6.856s, whereas the execution time of GPURAS is 0.017s. When GPURASMC uses 60% of all points, the execution time is 0.011s. When the magnitude is increased by three times (1024*1024~$10^6$), the execution time of CPUMC did not terminate even after more than 10 minutes. However, the execution time is 0.035s for GPURAS and 0.032s for GPURASMC uses 60% of all points. The efficiency of GPURAS and GPURASMC is improved by hundreds of times compared to the CPU algorithm.

6. Conclusions

In this paper, a novel and efficient polygon intersection area calculation algorithm is proposed, which makes mild compromises in accuracy for better overall performances. Comparing and analyzing the relative area error rate of our algorithms and CPU-based deterministic algorithm, and the efficiency of our algorithms and CPU-based Monte Carlo algorithm. Experimental results showed that our algorithms are not affected by the number of polygons, only sensitive to the resolution. With the increase of the resolution, the error decreases but the time overhead is increased. Our algorithms are suitable for arbitrary complex polygons. Future work includes 3D spatial location relation determination of polygons.
Acknowledgment

The authors declare that there is no conflict of interest regarding the publication of this paper. This research was financially supported by Chinese National Defense Foundation of Science and Technology (Grant No. 3602027) and Jiangsu Province Science Foundation for Youths (Grant No.BK20150722).

References


