A Region Merging Algorithm for Radar Net Power Range Display Fusing

Liang Wang\textsuperscript{1,a}, Hailong Ma\textsuperscript{1,b}, Xiao Wang\textsuperscript{2,c}, Yanze Qu\textsuperscript{1,d}, Ming Mao\textsuperscript{1,e}

\textsuperscript{1}PLA Strategic Support Force Information Engineering University Zhengzhou, China
\textsuperscript{2}PLA unit 66135 BeiJing, China

\textsuperscript{a}1747207681@qq.com, \textsuperscript{b}mhl@mail.ndsc.com.cn, \textsuperscript{c}18256951646@163.com,
\textsuperscript{d}1134800814@qq.com, \textsuperscript{e}maoming12345@163.com

Abstract. Radar net power range display fusing is a common requirement of electronic warfare simulation. Though GDI+ and other third-party tools could be used to solve the problems of region merging effectively, there are still some defects like uncontrollable calculation process and results. To solve these problems, in this paper we proposed one region merging algorithm based on the polygon boolean operation principle. Firstly, we introduce four key steps of the algorithm: adjusting clockwise order, computing intersection, extracting the envelope and region merging. Secondly, the data structure of the algorithm is presented. Finally, to verify the algorithm validity, we use some radar network scenarios as samples and analyze the algorithm performance briefly. The results show that when the radar number is less than 50 and the drawing interval is bigger than 15 ° , this algorithm can meet the demand of radar net power range display fusing in semi-real time and real time simulation.

Keywords: Radar net, power range, region merging, polygon Boolean operation, GDI+.

1. Introduction
In warfare simulation, to fulfill comprehensive statistics\cite{1}, simplified interface\cite{2} and other operational requirements, military personnel need to integrate the power range of multi-radar network from multiple independent effectiveness areas into one single effectiveness area in display. To solve the problem of radar net power range display fusing, the "point-by-point calculation" algorithm was proposed in paper\cite{3}, the "eraser" algorithm was proposed in paper\cite{4}, and the "grid segmentation and screening" algorithm was proposed in paper\cite{5}. Most of these methods were simplified technically according to the specific application scenarios, however, when the accuracy requirements were improved, the efficiency of the algorithm would decrease obviously. Radar net power range fusing, essentially a polygonal region merging problem in computer graphics, means seeking for the union of different areas. This problem has arisen for long time, and to solve it, the GDI+, namely graphics device interface of Windows for programming, has already embedded one effective union algorithm in the Region class\cite{6}. Besides, it can display the combined area, but is unable to get the specific boundary point data, causing some obstacles in boundary display, area calculation and other requirements. In this paper, we implemented a region merging algorithm for radar network power range display fusing based on some existing...
algorithm theories such as paper[7~9] to provide some reference for related warfare-operation researchers.

2. Problem Description

At present, the warfare simulation based on VC++ programming language mostly adopts algorithm embedded in GDI+ to realize the radar net power range display fusing. However, in the relevant known applications and results, the algorithm embedded in GDI+ fails to obtain the boundary points of the syncretic region, and it even has calculation errors when dealing with complex situations. Therefore, it is necessary to seek for a more effective region merging algorithm. The problem can be investigated by following the next three steps.

2.1. Radar power range display

The radar power range refers to the closed area surrounded by the boundary point of the maximum operating range of the radar on each position, which can also be regarded as its field of view. In warfare simulation, the radar power range is usually sampled more than one time at fixed angle intervals, and the radar operating range boundary points are calculated respectively. Then, the radar power range is approximated by the polygon formed by these sampling boundary points. Fig.1 shows three simulation examples of radar power range under ideal conditions.

![Fig. 1 Simulation example of radar power range](image)

Among them, Fig.1(a) and Fig.1(b) are the approximate maps of radar power range obtained by sampling the 0~360° radar field of view with intervals of 30° and 10° respectively; Fig.1(c) is the approximate map of radar power range obtained by sampling the 30~150° radar field of view with intervals of 10°. It can be seen from the figure that:

1) The smaller the sampling interval is, the less distortion the radar power range map approximated by sampling will be. In practice, a better display effect can be achieved when the sampling interval is set to 5°; 
2) When the radar field of view cannot cover the whole direction (0~360°), the deployment position of radar should also be connected in the drawing apart from the boundary point of radar action range.

2.2. Radar power range display under jamming conditions

When the radar is jammed, its operating range on some positions will be compressed and reduced, which can be calculated by the following formula [10].

$$R_i = \sqrt{\frac{K_j P_i G_j^2 \sigma}{4\pi \sum_i \frac{P_j G_j G_i (\theta_i)}{R_{ji}^2 \cdot L_j}}}$$

(1)
In the equation, “$R$” denotes radar power range under interference on azimuth $\phi$ (unit:m), “$K_j$” denotes the interference suppression coefficient required by the rule of power (unit: times), “$P_i$” denotes radar transmission power (unit:W), “$G_i$” denotes main lobe gain of radar antenna (unit:times), “$\sigma$” denotes the RCS of target(unit:m$^2$), “$P_{ji}$” denotes transmission power of jammer i (unit:W), “$G_{ji}$” denotes antenna gain of jammer i (unit:times), “$R_{ji}$” denotes the distance between jammer i and radar(unit:M), “$L_j$” denotes integrated power transmission losses of jammer i, “$\theta_i$” denotes the intersection angle between jammer i’s position relative to the radar location and azimuth $\phi$ (unit: °), and “$G_i(\theta_i)$” denotes the receive gain in the orientation of jammer i of radar antenna. $G_i(\theta_i)$ could be calculated by the following equation:

$$G_i(\theta) = \begin{cases} 
G_i & 0 \leq \theta \leq \frac{\theta_{0.5}}{2} \\
K \left(\frac{\theta_{0.5}}{\theta}\right)^2 G_i & \frac{\theta_{0.5}}{2} < \theta < 90^\circ \\
K \left(\frac{\theta_{0.5}}{90}\right)^2 G_i & \theta \geq 90^\circ
\end{cases}$$

In the equation, “$K$” is the directivity constant of radar antenna, and “$\theta_{0.5}$” denotes the main lobe width of radar antenna (unit:°).

Let $P_i=10^6$, $G_i=10^4$, $\sigma=5$, $P_{ji}=10^3$, $G_{ji}=10^3$, $L_j=100$, $K_j=10$, $K=0.02$, $\theta_{0.5}=30$, and Fig. 2 shows three simulation samples of radar power range under jamming conditions.

Fig. 2 Simulation examples of radar power range under jamming conditions

2.3. Radar net power range display fusing

When two radars are deployed in close proximity, their power range will overlap in some areas. To achieve better display fusing of radar net power range, it is necessary to display only the outer envelope of all radar’s power range in the radar net, but not the single radar power range contained in the outer envelope.

The Union function in the Region class of GDI+ provides an implementation algorithm for region merging, and it could be used in the following way.

1) Add GDI+ declaration to stdafx.h function:

```cpp
#pragma comment(lib,"gdiplus.lib")
#include <gdiplus.h>
using namespace Gdiplus;
```

2) Add GDI+ flag variable to CWinApp class:
3) Add GDI+ startup code to initInstance function of CWinApp class:

```cpp
gdiplusStartupInput gdiplusStartupInput;
gdiplus::GdiplusStartup(&m_gdiplusToken, &gdiplusStartupInput, NULL);
```

4) Overload the ExitInstance function of CWinApp class, and add the following GDI+ close code:

```cpp
Gdiplus::GdiplusShutdown(m_gdiplusToken);
```

5) Add Region variables “rgnRadar” and “rgnNet” to the radar class and the radar net class respectively, which are used to store the power range of a single radar and the syncretic power range of the radar net.

6) Add a GraphicsPath variable rPath to the radar class, which is used to store the power range envelope of each radar entity.

7) For each radar entity, the AddPolygon function of GraphiSpath class is called respectively to load the radar power range envelope into rPath, and then the Complete function of Region class is called to initialize rgnRadar with rPath.

8) The Union function of Region class is called to merge the rgnNet of radar net with the rgnRadar of each radar entity successively;

9) Add the following area drawing code into the OnDraw function of CView class:

```cpp
Graphics Graphics(PDC->m_hDC);//declare and initialize the Graphics tool.
SolidBrush brush(Color(25, 255, 0, 0));//Use the red brush.
graphics.FillRegion(&brush,&rgnUnioned);// Filling area.
```

It can be seen from the figure:

1) As Fig. 3(a) and Fig. 3(b) show, in most cases, GDI+ can meet the requirements for display fusing of radar net power range. However, as shown in Fig. 3(c), there are errors in display fusing calculation.

2) GDI+ can present the syncretic power range of radar net in form of filled region, but it does not arrive at the boundary envelope of the syncretic region. By inquiring GDI+ manual, the GetData function in the Region class is available for getting regional boundaries data, but the detailed content and format of data are not discussed in the manual, namely the GDI+ can only provide visual display effect of radar net display fusing in the form of area filling, but does not support the drawing of envelope of the syncretic zone map. Furthermore, it doesn’t support high-level application requirements like area computation relying on the envelope data.

In view of the above reasons, we decided to independently implement a region merging algorithm for radar net power range display fusing by referring to the basic principle of polygon Boolean operation.

### 3. Region Merging Algorithm

Display fusing of radar net power range, requires that for each single radar net power range area, all inner envelope contained in other envelope area of radar net power will be removed. Then the remaining envelope will be joined together in sequence. For this question, the essence is to merge multiple simple regions.
regions, where the "simple area" refers to the area having no cross between its each side, including convex area and concave area.

There are only three possible relationships between two simple regions: non-intersection, intersection and inclusion, where non-intersection means that no nodes of one region is contained in the other region, intersection means that either region has some but not all nodes in the other region, inclusion means that all nodes in one region are contained in the other region. It is relatively simple to merge regions with non-intersection or inclusion relations: for the former, it means to duplicate records of data from two regions, while for the latter, it means the larger region itself. However, merging intersected regions is more complex. For each region, it is necessary to find out all points that are not included by the other region, as well as the intersection points between regions, which are then connected successively to form a closed loop envelope.

Generally speaking, merging two simple regions requires three steps: adjusting the clockwise order, intersection point calculation and extracting the outer envelope.

3.1. Adjust the clockwise order
To avoid confusion in regional merging results, it is necessary to make all regions have the same directivity first. There are two kinds of directivity for a simple region: clockwise and counterclockwise. In this paper, clockwise sequence is selected as the criterion of directivity in region merging.

The key of adjusting the clockwise order is to judge the order of nodes in regions. If the region is already formed in clockwise order, there is no need to change it. However, if the region is counterclockwise, it is necessary to change the order of its nodes to reverse order. With the Y-axis downward as positive, in the screen coordinate system, the way to judge the directionality of a region is as follows. Firstly, find the node in the upper left corner of a region, which is denoted as \( \text{P} \). Secondly, find the former and next node of \( \text{P} \), and denote them as \( \text{Q} \) and \( \text{M} \) respectively. Finally, judge whether the results of the vector product is positive or not. If it is positive, the directivity of area is clockwise; otherwise, it is counterclockwise.

3.2. Intersection point calculation
When merging regions, since we do not know which edges will intersect in advance, we need to traverse all edges of each region and calculate the intersection points for all possible intersecting edges. Denote one edge of area A as PQ, and one edge of area B as MN, and then:

1) when the slopes of edge PQ and edge MN exist and they are not equal, these two edges will intersect at point \((x, y)\) as calculated by the formula below:

\[
\begin{align*}
    x &= \frac{y_p - y_M - k_A x_p + k_B x_M}{k_B - k_A} \\
    y &= k_A \left( x - x_p \right) + y_p \\
    k_A &= \frac{y_Q - y_p}{x_Q - x_p} \\
    k_B &= \frac{y_N - y_M}{x_M - x_N}
\end{align*}
\]

If \( x \in \left[ \min (x_p, x_Q, x_M, x_N), \max (x_p, x_Q, x_M, x_N) \right] \) and \( y \in \left[ \min (y_p, y_Q, y_M, y_N), \max (y_p, y_Q, y_M, y_N) \right] \) then edge PQ is considered to intersect with edge MN, otherwise the two sides will not intersect;

2) if the slopes of both sides do not exist, or if they are equal, the two sides are considered non-intersected;

3) if only one slope of edge does not exist, say, the edge for PQ, then the intersection point of these two edges is calculated as follows:
\[
\begin{align*}
x &= x_P \\
y &= \frac{y_N - y_M}{x_N - x_M} (x - x_M) + y_M
\end{align*}
\]

If and only if \[ y \in \left[ \min(y_P, y_Q, y_M, y_N), \max(y_P, y_Q, y_M, y_N) \right] \] the two sides are considered intersecting.

Note that if the intersection point happens to be one of the original four vertices, a special flag needs to be set for the convenience of subsequent processing.

3.3. Extract the outer envelope

The basic idea of extracting the outer envelope is as follows: start from the first node not in the other region and connect the next node successively; when the intersection point is encountered, transfer to the other region or continue to stay in this region according to the nature of the intersection point until the process returns to the starting point. Holes may occur when concave areas are merged, where all nodes not connected need to be checked. Repeat the process for nodes that are not in the other region to obtain all the holes. Fig. 4 provides a brief demonstration of this principle.

1) In Fig. 4(a), according to the algorithm, we extract the outer envelope from point A_0 and connects points A_1 and A_2 successively; since point A_2 is an ordinary intersection point, turn to its corresponding intersection point B_2 in area B, and continue to turn to area B for extraction. Then, after five extractions: B_3 (A_7), A_10 (B_4), B_7 (A_11), A_14 (B_10) and B_1 (A_15), a closed path is formed finally by returning to the original starting point A_0. At this time, the unconnected nodes are checked and A_4 is not found in the region B, so another envelope path needs to be extracted from the point A_4. Finally, the result of region merging of Fig. 4(a) is a closed region with holes.

2) In Fig. 4(b), when extracting outside envelope, the first intersection A_1 correspond to intersection B_2 in B area, which happens to be a original vertex of B area. Then, judge whether to transfer to B area or to stay in area A in the following way: check the next point of corresponding intersection (B_3) is in this area (area A) or not; stay in this area if yes and go to the other area otherwise.

In the above process, whether a point is within a certain region will be determined repeatedly. The method is as follows: lead a ray to the right (or any direction) from the point, and if the ray intersects with an even number of points in the specified region, then the point is not within the specified region; otherwise, it is within the region.

It should be noted that in theory, regional node coordinates are non-integers in most cases. However, because they are in pixels, their value will be forced to round when displayed on the screen, and the resulting rounding errors may sometimes lead to unforeseen mistakes, as shown in Figure 3(c). In this regard, we appropriately eased the requirements on the calculation accuracy of coordinate points, and
regarded the points with horizontal and vertical coordinate errors within 1 pixel as the same point. Thus, many significant errors were effectively avoided to basically meet the engineering application requirements of warfare situation display.

### 3.4. Multi-region merging

Since the merging result of two simple regions is not necessarily a single closed region, the nested operation of merging two regions cannot solve all the problems of multi-region merging. Similar to the two-region merging algorithm, the multi-region merging also requires three steps: adjusting the clockwise order, intersection point calculation and extracting the outer envelope. The difference lies in that all the polygon regions need to be calculated simultaneously during each step operation.

### 3.5. Data structure

According to the above algorithm principle analysis, some real-time data need to be recorded in the process of program calculation for subsequent operations. To store these real-time data, two structures are defined: sPolygon for polygon region and sPolygonPoint for region node. Specific definitions are as follows:

1) Definition of data structure sPolygon

```c
struct sPolygon
{
sPolygonPoint* ptFirst; // Pointer to the first node in the region
}
```

2) Definition of data structure sPolygonPoint

```c
struct sPolygonPoint
{
sPolygon* pPolygon; // Pointer to the region where the node is located
double x; // X coordinate
double y; // Y coordinate
CPolygonPoint* ptPre; // Previous node
CPolygonPoint* ptNext; // Next node
int nType; // node types: 0: normal original vertex of the region; 1: normal intersection; 2: original vertex of the region, but at the intersection
CPolygonPoint* ptCorrespondingPt;
// If the point is an intersection, then the pointer points to its corresponding intersection in the other region.
CPolygonPoint* pUnionedPolygon;
// If the point has been added to the fused region, then the pointer points to the syncretic region
}
```

### 4. Examples Analysis

#### 4.1. Examples of application

The algorithm proposed in this paper is used to re-calculate the three scenes in Fig. 3, and the results are shown in Fig. 5.

As can be seen from the figure, the algorithm proposed in this paper successfully obtains the boundary points of the syncretic region, which outperforms GDI+ in fulfilling the application requirements.

#### 4.2. Performance analysis

For further analyzing the algorithm performance proposed in this paper, in the context of Intel Xeon E5-1620v4 CPU, Win7 Service Pack 1 (SP1) environment, we randomly set the positions of radar and jammer, and studied how the radar number and the sampling interval of radar power range would impact time consumption of the algorithm. The results are shown in figure 6.
As can be seen from the figure 6:

1) As the number of radars increases, time consumption of the region merging algorithm gradually exhibits an exponential increasing trend;

2) Increasing the sampling interval of the power range tends to reduce the number of nodes in the region, thus greatly reducing time consumption of the algorithm.

Fig. 7 shows the merging effect of regions with 20 radars and a sampling interval of 15°, which can basically meet the engineering level visualization requirements in warfare simulation.
When the sampling interval is 15° and the number of radars is 50, the time consumption of the algorithm is 2874ms, which is acceptable for the semi-real-time simulation with a step of 5s and the real-time simulation with a step of 10s. However, for the ultra-real-time simulation with a step of 1s or 0.1s, other design methods are needed to further improve the regional merging efficiency (Figure 7).

5. Conclusion
The display fusing of radar net power range is a common requirement in warfare simulation. To solve related problems, some targeted technical solutions have already been proposed, such as GDI+ of Windows and other third-party tools, but the problem of uncontrollable computing process still exists. The performance of the region merging algorithm proposed in this paper may not be optimal, but the principle and process are clear for beginners to realize independently, and also convenient for customized adjustment according to advanced application requirements. This is a unique advantage that other third-party tools do not have.

References
[1] WU Hai-tao, GAO Yuan. Discussion on Detection Range and Radar Number in Radar Network [J]. Fire Control and Command Control, 2014, 39(Suppl):32-35.
[2] Zhang Yunlei, Wang Dong, Xi Zemin, et al. GIS-based Radar Network Detection Efficiency Simulation [J]. Fire Control and Command Control, 2014, 39(7):39-42.
[3] WANG Shu-wen. A Drawing Method for Computer Simulation of Radar Network Plane Detection Range [J]. Journal of Air Force Radar Academy, 1999, 13(3):51-53.
[4] Zhou Qiming, Wang Zhongjie, Li Xia, et al. Complex Area Graph Processing Algorithm in Radar Network Power Simulation [J]. Modern Defense Technology, 2008, 36(1):87-91.
[5] Guo Lak, Zhang Yang, He Jun. Research on Visualization Method of Radar Network Power Range [J]. Fire Control and Command Control, 2012, 37(7):95-97.
[6] Mahesh Chand (Written), Han Jiang (Translated). GDI+ Graphic Programming [M]. 2005.
[7] Wu Yunxing. Boolean Operation of Polygon Based on Boundary Recognition [J]. Journal of Computer Aided Design & Graphics, 1994, (4):260-265.
[8] Chen Xuedong, Zhang Lichao, Huang Shuhuai. Boolean Operation of Polygon Based on "Interior Point" Recognition [J]. Computer Engineering and Science, 2005, 27(5):42-44.
[9] Qi Dongzhou, Wu Min. Efficient Boolean Calculation Method for Polygons [J]. Computer Application, 2014, 34(S2):78-82.
[10] Shao Guopei, Cao Zhiyao, He Jun, et al. Analysis of Electronic Countermeasure Combat Effectiveness [M]. PLA Publishing House [M], 1998.