International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 1
ISSN: 2088-5334

Methods for Software Visualization of Large Graph Data Structures

Velin Kralev #1, Radoslava Kraleva #2

Department of Informatics, South-West University "Neofit Rilski", 66 Ivan Michailov Str., Blagoevgrad, 2700, Bulgaria
E-mail: [1]velin_kralev@swu.bg, [2]rady_kraleva@swu.bg

***Abstract*— In this paper, three different methods for software visualization of large graph structures, respectively Rectangle, Intersection and Combined are presented. The basic concepts for using software development environments are outlined. Their capabilities for visual designing and event-oriented programming are discussed. A brief analysis of the basic features of the environment used to develop the ClipRect Monitor application is made. The main functions of this software are also presented. All experimental results in this study are generated with this application. According to the methodology, six graphs are prepared to determine the effectiveness of the three methods. The number of vertices and the edges of these graphs are proportional to the size of the drawing area (canvas). The drawing areas are also six and have different sizes, such that each subsequent area has a height and width twice the size of the previous one. Besides, for all areas, the width/height ratio is exactly 16:9. This ratio is widely used in monitors as well as laptops, mobile phones and tablets. The largest drawing area that the ClipRect Monitor application scanned during the experiments is 128 000 x 72 000 pixels. This scan is performed for graph G_6 with 1 415 vertices and 100 000 edges. The visualization area is diagonally positioned relative to the drawing area. For each visualization area, each of the three methods, respectively Rectangle, Intersection and Combined is performed. The Combined method executes the Rectangle method first and then the Intersection method. The results show that the Intersection method was the slowest compared to the other two methods in terms of the number of edges of the graph that are analyzed. When the visualization area is internal to the drawing area, the Rectangle method performs better than the Combined method. The Rectangle method gives the best result in terms of time for analysis and drawing of the edges of the graph. The Combined method combines the characteristics of the other two methods. This method is optimal in terms of the time of analysis of the need to draw the edges of the graph relative to the number of drawn edges.**

***Keywords*— graph; large graphs; graph structure; software development; software visualization.**

I. Introduction

Graph theory is a scientific field that has evolved very rapidly in recent decades [1]. Complicated real problems can be represented by graphs [2] and [3]. Other similar problems are related to finding the shortest routes [4] and generating university timetables [5]. Problems from other scientific fields, such as [6]–[10] can also be described and analyzed by graphs. The graph structures can be stored in databases and retrieved by web services [11].

The visual representation of graphs is the process by which different geometric objects - lines, circles, and regular polygons are drawn on a computer screen. Typically, the x and y coordinates of a given vertex are known, or the coordinates of the two ends of an edge are known, i.e., these are the coordinates of the two vertices that are incident to this edge. When visualizing multigraphs, parallel edges are drawn using Bezier curves to avoid overlapping lines.

When, in a graph, the total number of vertices and edges is small (for example, in the order of hundreds to thousands), the drawing of a graph by geometric objects is performed relatively quickly (i.e., imperceptible to the user). When increasing the number of vertices, respectively increasing the number of edges in a graph (up to hundreds of thousands, even millions), it is necessary to use methods to optimize the process of drawing the graph structure on the computer screen. Usually, large graph structures contain millions of objects (vertices and edges) and cover large areas, depending on the used units of measurement [12]. These areas are much larger than the size of a computer screen, i.e., much larger than the screen resolution. The distance between the two farthest vertices can be in the order of tens of thousands of pixels (depending on the real object represented by the graph structure).

For large graphs, one effective method is to draw from all elements of the graph (vertices and edges) only those that fall within the visible area of the screen [12]. This approach can be used when the area of the graph is much larger than the size of the computer screen. When drawing each vertex of a graph, it is checked whether its coordinates are internal to the visible area of the screen or not. Only those vertices of the graph that fall into the visible area of the screen are drawn, and the others are only checked. In this way, the number of checks on whether a graph vertex falls within the visible area of the screen or not exactly n (n is the number of vertices in the graph).

There is a significant difference in the visualization of the edges of the graph compared to the visualization of the vertices of the graph. It is necessary to check the coordinates of each edge (i.e., the coordinates of the vertices that are incident to that edge) and to consider three cases. First, whether both vertices incident with this edge falls within the visible area of the screen. Second, whether one of the vertices incidents with this edge falls within the visible area of the screen. Third, if neither of the two vertices incidents with this edge falls within the visible area of the screen, the given edge intersects the visible area of the screen [13].

Whether a vertex V with x, y coordinates fall into a rectangular area defined by the points A, B, C and D, respectively with coordinates $A(x_1,y_1)$, $B(x_2,y_1)$, $C(x_1,y_2)$, and $D(x_2,y_2)$ can be made with the following logical expression:

$$B = ((x \geq x_1) \text{ and } (x \leq x_2) \text{ and } (y \geq y_1) \text{ and } (y \leq y_2)) \tag{1}$$

If the logical expression B is true, then the vertex V with the coordinates x, y falls in the visible area of the screen. A detailed description of this problem is presented in [13].

Since the intersection of the visible area of the screen by an edge is a special case of the line-line intersection problem, it will be briefly discussed here. Different algorithms for a line segment clipping by the rectangle are presented in [14], [15], and [16]. Various approaches to improve the performance of these algorithms have also been developed - [17] and [18].

In the general case, to calculate the intersection of two lines a and b, respectively, determined by the points $A(x_1,y_1)$ and $B(x_2,y_2)$ for a, and the points $C(x_3,y_3)$ and $D(x_4, y_4)$ for b, the following equations can be used [13]:

$$t =(x_1–x_3)*(y_3–y_4) – (y_1–y_3)*(x_3–x_4) / \\ (x_1–x_2)*(y_3–y_4) – (y_1–y_2)*(x_3–x_4), \text{ thus} \tag{2}$$

$$x = x_1 + t*(x_2–x_1) \text{ and } y = y_1 + t*(y_2–y_1) \tag{3}$$

or

$$u =(x_1–x_3)*(y_1–y_2) – (y_1–y_3)*(x_1–x_2) / \\ (x_1–x_2)*(y_3–y_4) – (y_1–y_2)*(x_3–x_4), \text{ thus} \tag{4}$$

$$x = x_3 + u*(x_4–x_3) \text{ and } y = y_3 + u*(y_4–y_3) \tag{5}$$

If the lines a and b are parallel or coincident, then:

$$(x_1–x_2)*(y_3–y_4) – (y_1–y_2)*(x_3–x_4) = 0 \tag{6}$$

If the lines a and b have an intersection point, then:

$$0 \leq t \leq 1 \text{ and } 0 \leq u \leq 1, \text{ i.e., } t \in [0, 1] \text{ and } u \in [0, 1] \tag{7}$$

When one of the two lines is horizontal, for example line b, defined by the points $C(x_3,y_3)$ and $D(x_4,y_4)$, then the parameters t and u can be calculated as follows:

$$t = (– (y_1–y_3)*(x_3–x_4)) / d, \text{ and} \\ u = ((x_1–x_3)*(y_1–y_2) – (y_1–y_3)*(x_1–x_2)) / d, \text{ where} \\ d = – (y_1–y_2)*(x_3–x_4), \text{ and } d \neq 0 \\ \text{if } 0 \leq t \leq 1 \text{ and } 0 \leq u \leq 1, \text{ i.e.} t \in [0, 1] \text{ and } u \in [0, 1], \text{ then} \\ x = x_1 + t*(x_2–x_1); y = y_1 + t*(y_2–y_1), \text{ or} \\ x = x_3 + u*(x_4–x_3); y = y_3 + u*(y_4–y_3) = y_3, \\ \text{because } y_3 = y_4 \text{ and } (y_4–y_3) = 0 \tag{8}$$

When one of the two lines is vertical, for example the line b, defined by the points $C(x_3,y_3)$ and $D(x_4,y_4)$, then the parameters t and u can be calculated as follows:

$$t = (x_1–x_3)*(y_3–y_4) / d, \text{ and} \\ u = ((x_1–x_3)*(y_1–y_2) – (y_1–y_3)*(x_1–x_2)) / d, \text{ where} \\ d = (x_1–x_2)*(y_3–y_4), \text{ and } d \neq 0 \\ \text{if } 0 \leq t \leq 1 \text{ and } 0 \leq u \leq 1, \text{ i.e.} t \in [0, 1] \text{ and } u \in [0, 1], \text{ then} \\ x = x_1 + t*(x_2–x_1); y = y_1 + t*(y_2–y_1), \text{ or} \\ x = x_3 + u*(x_4–x_3) = x_3, \text{ because} \\ y_3 = y_4 \text{ and } (y_4–y_3) = 0; y = y_3 + u*(y_4–y_3) \tag{9}$$

Although only the edges that intersect the visible area of the screen (or are internal to it) are drawn, it is necessary to check all the edges of the graph. Similarly, as it is necessary to check all n vertices of a graph, so it is also necessary to check all m edges of a graph.

In the present study, three different visualization methods for large graph structures will be analyzed. They will be experimentally verified.

The first method (Rectangle) checks that the rectangle of the visible area of the screen has a common area with the rectangle formed by the coordinates of the two vertices incident to the edge considered. In this method, all edges of the graph are checked, but only those in which the condition is fulfilled are drawn. The relative position of the two rectangles may be such that one rectangle is contained in the other, the two rectangles partially overlap or have no common area.

The second method (Intersection) checks whether or not an edge intersects the display area. Only when this is true, does a given edge draw. There are also three options. First, the edge is outside the visualization area. Second, the edge is inside the visualization area. Third, the edge intersects the visualization area at one or two points.

The third method (Combined) combines the Rectangle and Intersection methods. This method first checks whether the two rectangular zones (the one defined by the coordinates of the two vertices incident to the edge and the one defined by the visualization area) have a common area or not. When this is true, the second method is used to determine the coordinates of the point (or points) of the intersection of the edge and the visualization area. It should be noted that in the Combined method, the Intersection method may not be executed even if only one of the two points is internal to the visualization area. This is because in this case the edge must necessarily be drawn. Specialized software was developed to test and analyze the three methods.

There are many programming languages used for developing software for different purposes [19]. RAD Studio is an integrated development environment for the rapid development of applications of various types - console, desktop, mobile and web-based. Embedded compilers can generate executable code for different operating systems, such as Windows (x86 and x64), OS X (32-bit only), iOS, and Android. This integrated environment for application development includes a wide range of tools. For instance, a source code editor, a form designer for both the VCL library and the FMX multi-platform library, an integrated debugger for all target platforms including mobile, source control, and many others. Additionally, this integrated environment offers many ready-made libraries of classes and component packages. It also makes it possible to create new libraries of classes and components or to add ones developed by other developers [20].

The RAD Studio C++ Builder variant, which is based on the C++ programming language, will be used for this development. Therefore, some key concepts in this programming language will be presented.

C++ is one of the most used languages for developing professional software products [21]. This may be due to the flexible syntax that this programming language offers. It must be noted that other high-level programming languages offer this option as well, e.g., C# and Delphi. However, the C++ language has very good possibilities for structural and object-oriented programming. Many mechanisms for linking different program elements such as functions, objects, structural data types, recursive function calls, portability, and many others are also available.

The key concept of using C++ is the ability to use classes and objects. Classes allow data encapsulation, implicit type conversion, memory management, and more. When an application is running, it is possible to determine dynamically which object with which class (even parental) is associated with. This approach is known as polymorphism and is possible due to the "late bonding" technology. Application development environments that support C ++ usually include extensions that make it a "dialect," but they give developers additional capabilities that are not provided in the basic language standard. Such features include, for example, built-in functions, dynamic memory management, using aliases, reflections, and more [22]–[25].

The C++ Builder IDE (part of the RAD Studio package) is an environment for developing applications for different operating systems. Also, the integrated environment provides many tools and capabilities, such as expanding the VCL and FMX class libraries.

This IDE includes a powerful code editor with debugging capabilities, a designer of forms for designing applications with a graphical user interface (both for the VCL library and for the FMX multi-platform library); a debugger for all target platforms (including mobile and web-based). It also offers many ready-made libraries of component and data control, but also provides the ability to extend existing libraries by installing packages with components and modules with classes created by other developers.

The text editor of the environment has all the features that modern code editors offer. The editor supports a complete code function. This feature makes writing a code easier, it also reduces the chances to commit syntax errors. When certain system or user events arise when a user interacts with controls from the user interface of an application, it is necessary to write program code in response to these actions. This code is encapsulated in functions called event handlers. The implementation of these functions is done in the code editor. The event handler function is the application response to the event that occurred.

The visual design tool, i.e., the Form Designer, makes it possible to create a quick (and relatively easy) graphical user interface for an application. This designer can be used both for designing a graphical user interface for form-based applications, as well as for designing mobile applications and web pages. The designer also supports real-time data visualization technology (from different sources). This enables developers to "see" the final view of an application during the design time stage.

The final stage in developing an application is its compiling into an intermediate language or machine instructions. This is necessary to enable either the virtual machine or the target operating system to start an application. In RAD Studio (C++ Builder) there are two ways to compile an application. First, compiling with subsequent testing and second, compiling, and testing simultaneously (using the built-in debugger). The testing process enables developers to check the values of the variables and the results of the functions during the runtime of an application.

II. Materials and Method

ClipRectMonitor software is created for experimental purposes. A session with the ClipRect Monitor application is shown in Fig. 1.

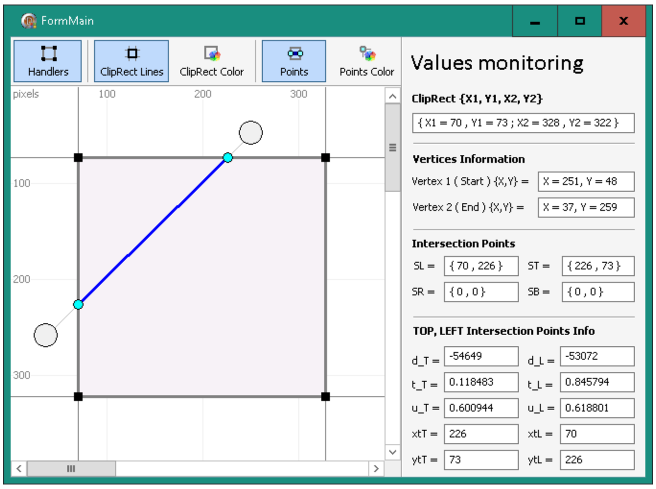

Fig. 1 A session with the ClipRect Monitor application

The ClipRect Monitor application has the following functionalities:

- Setting the coordinates of the visualization area by typing on the keyboard or using the mouse with the manipulators to adjust the size;
- Setting the coordinates of the two vertices that are incident to the edge (possible by entering the coordinates of the vertices from the keyboard or dragging the vertices with the mouse);
- Calculating the coordinates of the points of intersection of the edge with the visualization area;
- Calculating the parameters needed to calculate the coordinates of the intersection points of the edge with the visualization area (as presented in I. Introduction).

All parameters are displayed by the ClipRect Monitor application in a special panel - Values Monitoring, which is shown in Fig. 2.

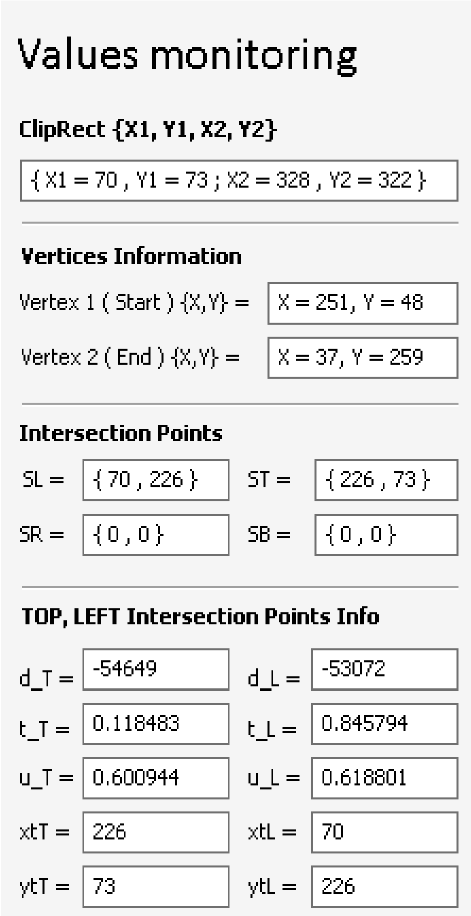

Fig. 2 The Values Monitoring Panel

Since each edge can have 0, 1, or 2 intersection points with the visualization area, no more than two sets of controls are needed to visualize the parameters used to calculate the coordinates of the intersection point (or points). Thus, the possible options for calculating the parameters are exactly eleven, respectively:

- Without intersection point;
- Intersecting one of the borders of the rectangular visualization area - Left, Top, Right or Bottom;
- Combinations of the intersection of two borders of the rectangular area, respectively Left - Top, Left - Right, Left - Bottom, Top - Right, Top - Bottom and Right - Bottom.

The ClipRect Monitor application has the function of creating and storing (in external files) graph structures with many vertices and edges (in the order of millions). Each graph can be visualized by drawing the vertices and edges (Fig. 3), by drawing the vertices and rectangular areas formed by the two incident vertices with each edge (Fig. 4), and combined by drawing the vertices, edges and rectangular areas formed by the two vertices incident to each edge. The coordinates of the vertices incident to each edge represent the coordinates of the upper left corner and the lower right corner of each rectangular area in which the given edge is inscribed.

A graph presents these three methods for visualization with 15 vertices and 105 edges (Figs. 3, 4 and 5).

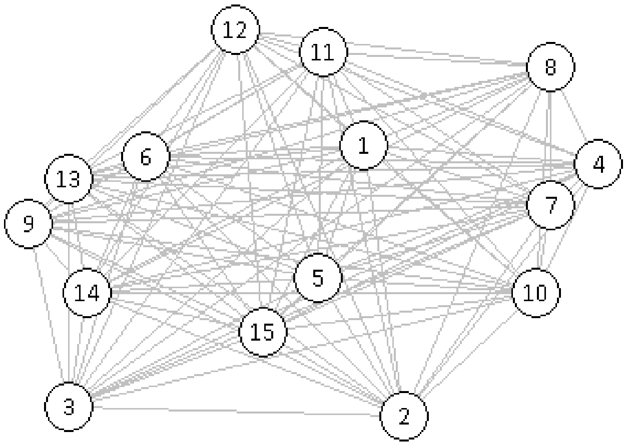

Fig. 3 Grpah G_15_105 visualized by vertices and edges

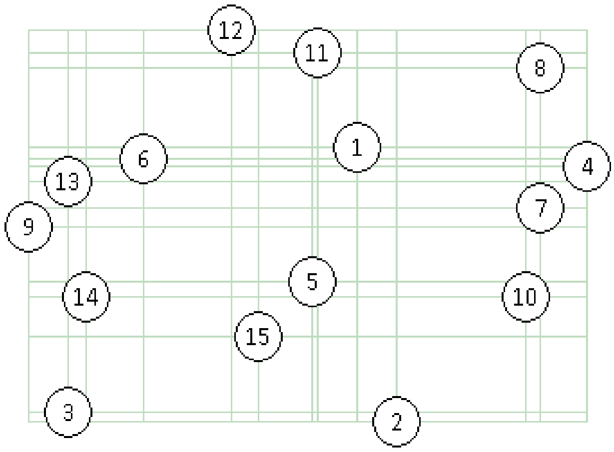

Fig. 4 Graph G_15_105 visualized by vertices and rectangles

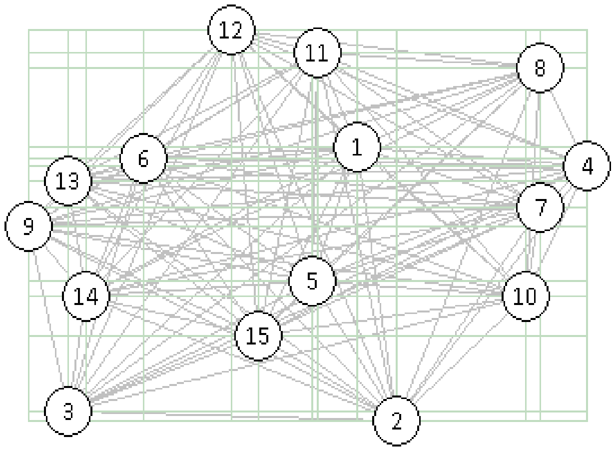

Fig. 5 Graph G_15_105 visualized by vertices, edges, and rectangles

The graph structures are stored internally in dynamic arrays for the vertices and edges, respectively. The ClipRect Monitor application can process these arrays. Each dynamic array is a record type structure (i.e., they are record arrays). Each record is a collection of values of a data. Because dynamic arrays are read-write, any value can be read and written by the ClipRect Monitor app.

III. Results and Discussion

The experiments in this study were conducted with the ClipRect Monitor software. This application was run on a computer with the Windows 10 operating system (OS).

A. The Methodology of the Experiments

Six graphs were prepared to determine the effectiveness of the three methods. The number of the vertices and the edges of these graphs was proportional to the size of the drawing area – canvas (Table I). The drawing areas are also six and have different sizes, such that each subsequent area has a height and width twice the size of the previous one. Besides, for all areas, the width/height ratio is exactly 16:9. This ratio corresponds to the standard Full HD (Full High Definition). This standard is widely used in monitors as well as laptops, mobile phones, and tablets.

TABLE I
Summarized Information of Graphs and Canvases

Graph Information			**Drawing Canvas Information (in px)**			
ID	**Vertices**	**Edges**	**Width**	**Height**	**W/H**	**Canvas Area**
G_1	250	3 125	4 000	2 250	1.78	9 000 000
G_2	355	6 250	8 000	4 500	1.78	36 000 000
G_3	500	12 500	16 000	9 000	1.78	144 000 000
G_4	710	25 000	32 000	18 000	1.78	576 000 000
G_5	1 000	50 000	64 000	36 000	1.78	2 304 000 000
G_6	1 415	100 000	128 000	72 000	1.78	9 216 000 000

The ClipRect Monitor application was executed four times for each of the six graphs. The application scanned the drawing area diagonally and counted the time needed to analyze and draw the vertices and edges of the graph that fall into the corresponding visualization area. This operation was performed for each of the three methods (Rectangle, Intersection, and Combined) to determine the objects to be visualized for each specific area. The results obtained were averaged to a more accurate account for the execution time of each method.

B. Experimental Conditions

The ClipRect Monitor application was run on a personal computer (PC) with 64-bit Windows 10 OS (Professional). The hardware configuration has the following characteristics: Processor: Intel Core i5-9300H (four cores, eight logical processors with 2.40 GHz base frequency (4.10 GHz max frequency), 8MB Cache; RAM Memory: 8 GB.

C. Experimental Results

Table II shows the results of the experiment conducted with graph G_6 (containing 1415 vertices and 100000 edges, respectively). The results of the other graphs G_1 ÷ G_5 are similar. All graphs were tested with 100 visualization areas.

The largest drawing area that the ClipRect Monitor application scanned during the experiments was 128 000 x 72 000 pixels. This scan was performed for graph G_6 with 1 415 vertices and 100 000 edges, respectively. With the three methods (Rectangle, Intersection and Combined), the visualization area was diagonally positioned relative to the drawing area.

TABLE II
Results for the G_6 Graph (From 1 280 x 729 Position)

Clip Rectangle Position		**Drawing Method**					
		Rectangle		**Intersection**		**Combined**	
Left	***Top***	***Edges***	***Time***	***Edges***	***Time***	***Edges***	***Time***
0	0	11	547	24	2 110	11	531
1 280	720	50	547	62	2 141	39	547
2 560	1 440	121	547	93	2 172	64	547
3 840	2 160	341	547	136	2 157	95	547
5 120	2 880	545	547	177	2 172	140	547
6 400	3 600	846	547	271	2 172	211	547
7 680	4 320	1 443	547	342	2 156	265	562
8 960	5 040	2 083	547	389	2 171	312	578
10 240	5 760	2 435	547	426	2 156	341	593
11 520	6 480	2 838	547	457	2 187	370	594
12 800	7 200	3 307	547	480	2 156	377	609
14 080	7 920	3 988	562	570	2 187	463	609
15 360	8 640	4 607	563	625	2 172	514	640
16 640	9 360	5 340	563	810	2 172	706	641
17 920	10 080	6 099	563	885	2 172	781	672
19 200	10 800	6 799	562	899	2 157	796	688
20 480	11 520	7 563	578	905	2 141	788	688
21 760	12 240	8 356	578	932	2 157	810	719
23 040	12 960	9 184	562	929	2 172	815	734
24 320	13 680	9 789	578	940	2 172	835	734
25 600	14 400	10 703	562	1 055	2 157	952	765
26 880	15 120	11 226	578	1 070	2 172	981	766
28 160	15 840	12 125	578	1 085	2 156	993	797
29 440	16 560	13 070	579	1 137	2 156	1 049	812
30 720	17 280	13 909	594	1 221	2 172	1 147	844
32 000	18 000	14 771	578	1 197	2 156	1 127	844
33 280	18 720	15 622	594	1 200	2 156	1 137	875
34 560	19 440	16 419	593	1 264	2 172	1 187	891
35 840	20 160	16 944	594	1 226	2 157	1 163	891
37 120	20 880	17 864	594	1 293	2 156	1 241	922
38 400	21 600	18 513	594	1 298	2 172	1 238	922
39 680	22 320	19 198	625	1 325	2 172	1 278	938
40 960	23 040	19 386	594	1 300	2 156	1 260	954
42 240	23 760	20 409	594	1 325	2 157	1 280	969
43 520	24 480	21 262	610	1 422	2 156	1 373	1 000
44 800	25 200	21 692	610	1 434	2 140	1 399	9 84
46 080	25 920	22 059	610	1 446	2 172	1 414	1 016
47 360	26 640	22 686	610	1 447	2 156	1 419	1 015
48 640	27 360	23 119	594	1 488	2 172	1 461	1 015
49 920	28 080	23 511	609	1 429	2 156	1 418	1 032
51 200	28 800	23 879	609	1 491	2 172	1 483	1 047
52 480	29 520	24 458	610	1 504	2 188	1 493	1 047
53 760	30 240	24 801	609	1 535	2 156	1 528	1 062
55 040	30 960	25 177	609	1 544	2 172	1 531	1 063
56 320	31 680	25 224	609	1 577	2 172	1 569	1 079
57 600	32 400	25 747	610	1 568	2 172	1 564	1 078
58 880	33 120	25 818	609	1 589	2 172	1 585	1 078
60 160	33 840	26 162	609	1 557	2 172	1 557	1 093
61 440	34 560	26 371	610	1 604	2 156	1 603	1 093
62 720	35 280	25 939	625	1 588	2 156	1 588	1 094
64 000	36 000	26 124	625	1 529	2 172	1 529	1 078
65 280	36 720	26 073	609	1 500	2 156	1 500	1 094
66 560	37 440	25 941	610	1 554	2 156	1 554	1 078
67 840	38 160	25 769	609	1 510	2 172	1 506	1 078
69 120	38 880	25 788	625	1 565	2 172	1 563	1 078
70 400	39 600	25 450	625	1 448	2 156	1 444	1 078
71 680	40 320	25 264	625	1 432	2 172	1 425	1 079
72 960	41 040	24 697	610	1 495	2 172	1 487	1 063
74 240	41 760	24 669	609	1 484	2 156	1 477	1 063

TABLE III

RESULTS FOR THE G_6 GRAPH (FROM 75 520 X 42 480 POSITION)

Clip Rectangle Position		Drawing Method					
		Rectangle		Intersection		Combined	
Left	*Top*	*Edges*	*Time*	*Edges*	*Time*	*Edges*	*Time*
75 520	42 480	24 237	609	1 431	2 156	1 424	1 046
76 800	43 200	24 008	609	1 516	2 172	1 494	1 047
78 080	43 920	23 403	610	1 448	2 172	1 429	1 031
79 360	44 640	23 379	609	1 493	2 156	1 473	1 031
80 640	45 360	22 199	609	1 342	2 157	1 322	1 000
81 920	46 080	21 970	610	1 362	2 172	1 320	1 016
83 200	46 800	20 932	593	1 350	2 156	1 311	984
84 480	47 520	20 663	594	1 370	2 125	1 330	969
85 760	48 240	20 123	609	1 418	2 156	1 372	953
87 040	48 960	19 875	609	1 424	2 172	1 363	953
88 320	49 680	18 729	594	1 329	2 141	1 263	937
89 600	50 400	18 204	594	1 325	2 172	1 263	922
90 880	51 120	17 420	594	1 357	2 156	1 286	906
92 160	51 840	17 087	594	1 370	2 156	1 288	891
93 440	52 560	16 066	578	1 295	2 172	1 227	875
94 720	53 280	15 604	593	1 287	2 188	1 225	891
96 000	54 000	14 667	593	1 275	2 157	1 197	844
97 280	54 720	13 635	578	1 205	2 156	1 111	828
98 560	55 440	12 591	578	1 161	2 172	1 070	813
99 840	56 160	11 804	578	1 114	2 157	1 026	781
101 120	56 880	11 309	578	1 134	2 156	1 044	781
102 400	57 600	10 700	578	1 147	2 172	1 047	766
103 680	58 320	9 734	578	1 151	2 172	1 052	750
104 960	59 040	8 856	578	1 071	2 203	942	735
106 240	59 760	8 385	578	1 039	2 172	914	719
107 520	60 480	7 909	563	991	2 156	877	719
108 800	61 200	7 037	563	1 013	2 172	878	688
110 080	61 920	6 470	562	992	2 171	868	672
111 360	62 640	5 439	563	827	2 187	715	656
112 640	63 360	4 596	562	832	2 156	707	640
113 920	64 080	3 956	563	734	2 172	612	625
115 200	64 800	3 365	562	705	2 172	597	610
116 480	65 520	2 873	563	652	2 172	544	593
117 760	66 240	2 074	547	578	2 157	470	578
119 040	66 960	1 667	562	568	2 171	456	578
120 320	67 680	1 266	563	406	2 172	299	563
121 600	68 400	959	547	302	2 172	219	562
122 880	69 120	665	547	247	2 172	200	547
124 160	69 840	584	547	234	2 172	186	547
125 440	70 560	84	547	84	2 172	55	546
126 720	71 280	23	547	34	2 187	23	547

For each visualization area, each of the visualization methods was checked, respectively, which vertices and which edges of the graph under consideration (in this case G_6) should be drawn and which not. The Combined method executes the Rectangle method first and then the Intersection method.

Tables II, III, and Fig. 6 show that the Intersection method was the slowest compared to the other two methods in terms of the number of edges of the graph that were analyzed. In this method, the time for analysis and drawing of the corresponding edges of the graph is similar for all visualization areas. In the Rectangle and Combined methods, the time to analyze and draw the corresponding edges of the graph is similar when the visualization areas are at the beginning and end of the diagonal of the drawing area.

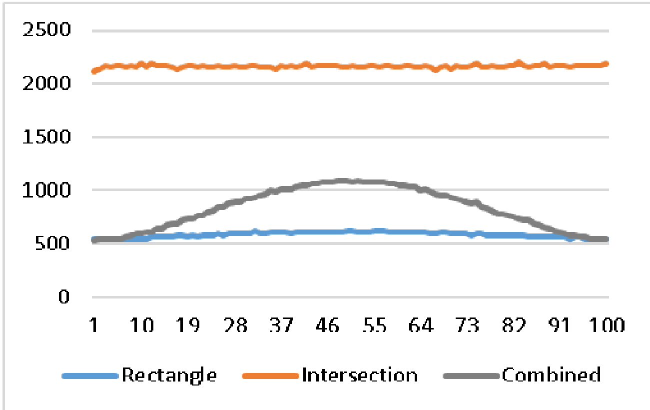

Fig. 6 Comparison between the three methods in terms of the time (*x*-axis in milliseconds) for analyzing and drawing the edges of a graph for each of the visualization areas (*y*-axis)

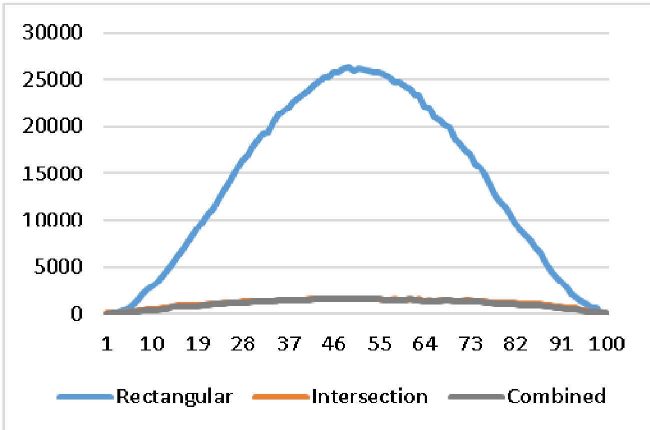

Fig. 7 Comparison between the three methods in terms of the number of analyzed edges (*y*-axis) for each of the visualization areas (*x*-axis)

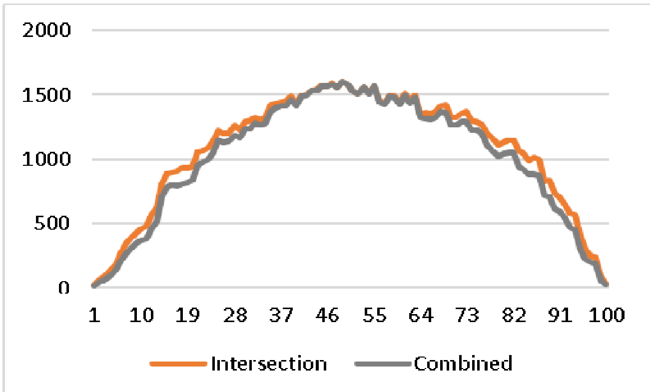

Fig. 8 Comparison between the Intersection and the Combined methods in terms of the number of analyzed edges (*y*-axis) for each of the visualization areas (*x*-axis)

When the visualization area is internal to the drawing area, the Rectangle method performs better than the Combined method. The Rectangle method gives the best result in terms of time for analysis and drawing of the edges of the graph. In addition, this time remains relatively constant for each of the visualization areas.

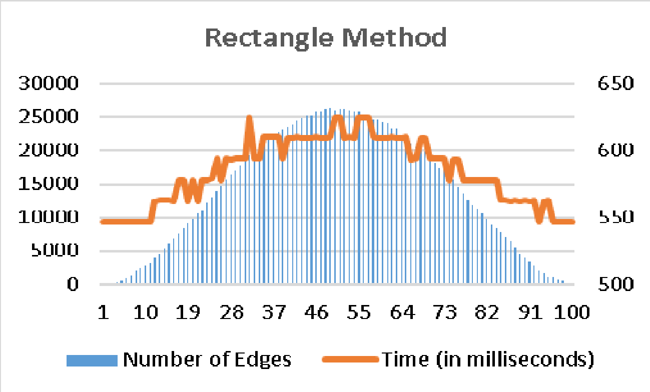

Fig. 9 Comparison between the number of drawn edges (left *y*-axis) and the time to analyze the need to draw those edges (right *y*-axis) for each of the visualization areas (*x*-axis) in the Rectangle method

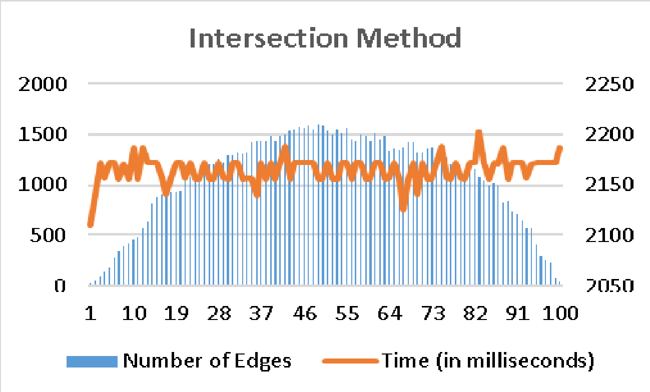

Fig. 10 Comparison between the number of drawn edges (left *y*-axis) and the time to analyze the need to draw those edges (right *y*-axis) for each of the visualization areas (*x*-axis) in the Intersection method

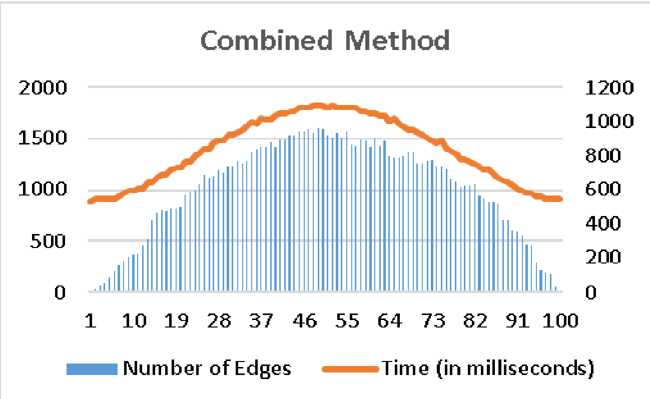

Fig. 11 Comparison between the number of drawn edges (left y-axis) and the time to analyze the need to draw those edges (right y-axis) for each of the visualization areas (x-axis) in the Combined method

Intersection and Combined methods analyze a smaller number of edges - only those that intersect the visualization area, but not those whose inscribed rectangles have a common area with the visualization area. However, the Rectangle method runs much faster in terms of time to analyze and draw the edges of a graph. This shows that the mathematical calculation of the intersection points that the Intersection and Combined methods performance is a much more time-consuming computational operation than the validation of the logical expression performed by the Rectangle method (see Equation 1).

Fig. 8 shows a comparison between the Intersection and Combined methods. These two methods both have almost identical results. This is because the Combined method, after executing the Rectangle method, also executes the Intersection method. However, the Combined method is preferred because, although with a slight difference, it performs better than the other two methods.

Fig. 9, Fig. 10 and Fig. 11 show summarized results of the three methods – Rectangle, Intersection and Combined, in terms of the number of drawn edges of the graph and the time to analyze the need to draw them for each of the visualization areas. Fig. 9 shows that in the Rectangle method, the time to analyze the need to draw the edges of a graph is proportional to the number of drawn edges. This is not true of the Intersection method (Fig. 10). In this method, the time to analyze the need to draw the edges of a graph is not proportional to the number of the drawn edges. This time remains relatively constant for all visualization areas. The third method (Combined) combines the characteristics of the other two methods. This method is optimal in terms of the time of analysis of the need to draw the edges of the graph relative to the number of the drawn edges.

IV. Conclusion

Three different methods for software visualization of large graph structures, respectively Rectangle, Intersection and Combined were presented in this paper. The basic concepts for using software development environments were outlined. Their capabilities for visual designing and event-oriented programming were discussed as well. A brief analysis of the basic features of the environment used to develop the ClipRect Monitor application was made. The main functions of this software were presented. All experimental results in this study were generated with this application. According to the methodology, six graphs were prepared to determine the effectiveness of the three methods. The number of vertices and the edges of these graphs were proportional to the size of the drawing area (canvas). The drawing areas were also six and had different sizes, such that each subsequent area had a height and width twice the size of the previous one. In addition, for all areas, the width/height ratio was exactly 16:9. This ratio is widely used in monitors as well as laptops, mobile phones and tablets. The largest drawing area that the ClipRect Monitor application scanned during the experiments was 128 000 x 72 000 pixels. This scan was performed for graph G_6 with 1 415 vertices and 100 000 edges. The visualization area was diagonally positioned relative to the drawing area. For each visualization area, each of the three methods, respectively Rectangle, Intersection and Combined was performed. The Combined method executes the Rectangle method first and then the Intersection method. The results showed that the Intersection method was the slowest compared to the other two methods in terms of the number of edges of the graph that were analyzed. When the visualization area was internal to the drawing area, the Rectangle method performed better than the Combined method. The Rectangle method gave the best result in terms of time for analysis and drawing of the edges of the graph. The Combined method combines the characteristics of the other two methods. This method is optimal in terms of the time of analysis of the need to draw the edges of the graph relative to the number of the drawn edges.

REFERENCES

[1] R. J. Wilson, *Introduction to Graph Theory*, 5th ed., Prentice Hall, 2010.

[2] M. Poobalaranjani, and R. Pichailakshmi, "Dominating cocoloring of graphs," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9(1), pp. 2545-2547, 2019.

[3] B. L. Natarajan, "Computation of chromatic numbers for new class of graphs and its applications," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8(8), pp. 396-400, 2019.

[4] V. Kralev, "Different applications of the genetic mutation operator for symmetric travelling salesman problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8(3), pp. 762-770, 2018.

[5] V. Kralev, R. Kraleva, and S. Kumar, "A modified event grouping based algorithm for the university course timetabling problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 9(1), pp. 229-235, 2019.

[6] B. Aleksandrov, A. Andreev, and N. Sinyagina, "Fast location of optimal separated routes in information exchange in computer networks," *International Conference Automatics and Informatics*, vol. 2, pp. 95-98, 2003.

[7] R. Kraleva, "ChilDiBu - A mobile application for Bulgarian Children with special educational needs," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 7(6), pp. 2085-2091, 2017.

[8] E, Ville, N. Sinyagina, and P. Borovska, "Deploying Trusted Computing," *Information Technologies and Controls*, pp. 28-32, 2009.

[9] J. Abawajy, A. V. Kelarev, M. Miller, and J. Ryan, "Distances of Centroid Sets in a Graph-Based Construction for Information Security Applications," *Mathematics in Computer Science*, vol. 9(2), pp. 127-137, 2015.

[10] A. K. Abdulsahib, and S. S. Kamaruddin, "Graph based text representation for document clustering," *Journal of Theoretical and Applied Information Technology*, vol. 76(1), pp. 1-13, 2015.

[11] V. Kralev, R. Kraleva, N. Sinyagina, P. Koprinkova-Hristova, and N. Bocheva, "An analysis of a web service based approach for experimental data sharing," *International Journal of Online Engineering*, vol. 14(9), pp. 19-34, 2018.

[12] V. S. Kralev, and R. S. Kraleva, "Visual analysis of actions performed with big graphs," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9(1), pp. 2740-2744, 2019.

[13] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics: Principles and Practice*, Addison-Wesley Professional, 3 ed., 2013.

[14] V.Skala, "O (lg N) line clipping algorithm in E^2," *Computers and Graphics*, vol. 18(4), pp. 517-524, 1994.

[15] V. Skala, "An efficient algorithm for line clipping by convex polygon," *Computers and Graphics*, vol. 17(4), pp. 417-421, 1993.

[16] V. Skala, "An efficient algorithm for line clipping by convex and non-convex polyhedra in E^3," *Computer Graphics Forum*, vol. 15(1), pp. 61-68, 1996.

[17] V. Skala, "A new approach to line and line segment clipping in homogeneous coordinates," *Visual Computer*, vol. 21(11), pp. 905-914, 2005.

[18] D. H. Bui, and V. Skala, "Fast algorithms for clipping lines and line segments in E^2," *Visual Computer*, vol. 14(1), pp. 31-37, 1998.

[19] J. Coenen, S. Gross, and N. Pinkwart, "Comparison of feedback strategies for supporting programming learning in integrated development environments (IDEs)," *Advances in Intelligent Systems and Computing*, vol. 629, pp. 72-83, 2018.

[20] K. Vassallo, L. Garg, V. Prakash, and K. Ramesh, "Contemporary technologies and methods for cross-platform application development," *Journal of Computational and Theoretical Nanoscience*, vol. 16(9), pp. 3854-3859, 2019.

[21] B. Stroustrup, *Programming: Principles and Practice Using C++*, Addison-Wesley Professional, 2 ed., 2014.

[22] P. Costanza, C. Herzeel, and W. Verachtert, "Comparing Ease of Programming in C++, Go, and Java for Implementing a Next-Generation Sequencing Tool," *Evolutionary Bioinformatics*, vol. 15, 2019.

[23] N. I. V'yukova, V. A. Galatenko, and S. V. Samborskii, "Support for Parallel and Concurrent Programming in C++," *Programming and Computer Software*, vol. 44(1), pp. 35-42, 2018.

[24] V. Dolgopolovas, T. Jevsikova, and V. Dagiene, "From Android games to coding in C – An approach to motivate novice engineering students to learn programming: A case study," *Computer Applications in Engineering Education*, vol. 26(1), pp. 75-90, 2018.

[25] D. Charousset, R. Hiesgen, and T. C. Schmidt, "Revisiting actor programming in C++," *Computer Languages, Systems and Structures*, vol. 45, pp. 105-131, 2016.