Comparison and Performance Evaluation of Boundary Fill and Flood Fill Algorithm

Bhawnesh Kumar, Umesh Kumar Tiwari, Santosh Kumar, Vikas Tomer, Jasmeet Kalra

Abstract*: In computer graphics, there are many polygon filling algorithms available like as inside and outside test, scan line method, boundary fill, flood fill, edge fill and fence fill algorithm. In this paper we focus on the flood fill and boundary fill algorithms for both four connected as well as eight connected pixels. This research paper shows the comparison and performance evaluation of boundary fill and flood fill algorithms with consideration of running time in C-language. And also shows the how stack affects the performance of the system due to overflow of buffer later on JAVA implementation with queue improves the system performance for large polygon. It also shows the literature review of various papers use the fill algorithm on different applications, some proposed the new and enhanced polygon fill algorithms.

***Index Terms*: Boundary fill, Flood fill, Four connected, Eight connected, Seed fill**

I. INTRODUCTION

In computer graphics, polygon comprises two things edge and vertex; it is closed shape in nature. It is created by the helps of lines. Polygons are two types first one convex and another one is concave. Polygon is said to be convex if line joining any 2-interior points of the polygon lies completely inside the polygon and non-convex polygons are known as concave. Further steps take place to fill the polygon with given color; here are so many algorithms in computer graphics used to fill the convex as well as concave polygons.

These algorithms[1] are inside and outside test, scan line[2][3], flood fill[4], boundary fill, edge fill[5] and fence fill. Each algorithm has own merits and demerits. They all algorithms implement on real world application somewhere some algorithm implemented successfully and somewhere they maybe some problems. If we want to fill any polygon by specific color then all filled pixels should be come inside the boundary of the polygon also identifying the interior pixels using inside and outside test.

These algorithms listed below and also represented in Figure 1.

1. Inside outside test
2. Scan line method
3. Boundary fill
4. Flood fill
5. Edge fill
6. Fence fill

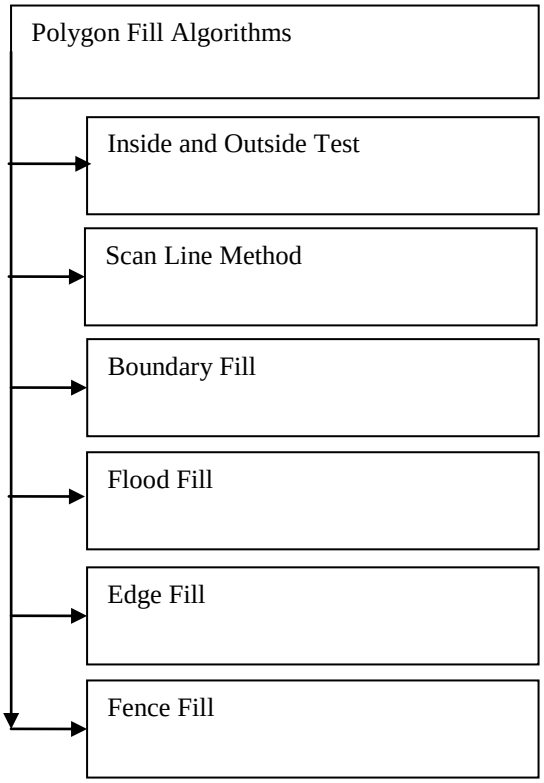

Fig. 1. Types of filling algorithms

II. LITERATURE REVIEW AND BACKGROUND

A. Literature Review

Many review paper, implementation paper, optimization and some others on real application [1][2][4][6][7] have been published on fill algorithms that show the comparative study on distinguish parameters. One of published paper[8] works for robotics agents using boundary fill algorithm. Another one paper[9] is used to count the regions with the help of efficient fill algorithm. In this paper [10] invented a novel approach for fill the polygon based on region using the proposed data structures of triples. Kallio[11] suggested a new technique to fill a two dimensional polygon using scan line flag and edge for anti-aliasing. This [12] method focuses on the seed fill algorithm with usage if internal connection of pixels of regions with a specified inner point.

New approach to fill the soft shadow has been proposed with help of image space flood fill algorithm[13].

Revised Manuscript Received on October 02, 2019.

Bhawnesh Kumar, Computer Science, D. A. V. College, Muzaffarnagar, India.

Umesh Kumar Tiwari, Computer Science & Engineering, Graphic Era Deemed to be University, Dehradun, India.

Santosh Kumar, Computer Science & Engineering, Graphic Era Deemed to be University, Dehradun, India

Vikas Tomer, Computer Science & Engineering, Graphic Era Deemed to be University, Dehradun, India

Jasmeet Kalra, Department of Mechanical Engineering, Graphic Era Hill University, Dehradun, India

Another one framework proposed [14] for an object retrieval using semi automatic method to improve the effectiveness using flood fill algorithm. In literature reviews, some give novel fill approaches or some use implement fill algorithm in real world application and others shows the comparative study.

B. Background

In case of both types of seed fill algorithms generally two methods are given to fill polygon pixel by pixel with neighborhood locations which are as follows:-

Four connected pixel positions:-

In this method, give pixel starting position from where filling algorithm will take place that position should be inside the polygon say (x, y). This position is also known as seed fill position, the neighbors of the (x, y) position are Right (x+1, y), Left (x-1, y), Up (x, y+1), Down (x, y-1) shows in figure 2.These are the four connected pixels position of given pixel inside the polygon.

	x, y+1 Up	
x-1, y Left	x, y Center	x+1, y Right
	x, y-1 Down	

Fig. 2. Four connected pixel positions

Eight connected pixel positions:-

In this method, give pixel starting position from where filling algorithm will take place that position should be inside the polygon say (x, y). This position is also known as seed fill position, the neighbors of the (x, y) position are Right (x+1, y), Left (x-1, y), Up (x, y+1), Down (x, y-1), Right-Up (x+1,y+1), Left-Up (x-1, y+1), Left-Down (x-1, y-1) and Right-Down (x+1,y-1) shows in figure 3.These are the eight connected pixels position of given pixel inside the polygon.

x-1, y+1 Left-Up	x, y+1 Up	x+1, y+1 Right-Up
x-1, y Left	x, y Center	x+1, y Right
x-1, y-1 Left-Down	x, y-1 Down	x+1, y-1 Right-Down

Fig. 3. Eight connected pixel positions

C. Boundary fill algorithm

As name indicates this algorithm works for boundary condition, it starts to fill the polygon from arbitrary pixel position with given fill color till the boundary color is encountered. In this algorithm the filled color and boundary should be different otherwise this algorithm fill not work properly. Boundary fill algorithm[15][6] can works both types of connected pixels four connected as well as eight connected. Below pseudo code is for four connected that can be same for the eight connected then four recursive statements should be added at the end.

Algorithm:

Step 1: Initialize the fill_color and boundary_color.
Step 2: find the current_value of position (x y).
Step 3: if current_value is not equal fill_color and current_value is not equal to boundary_color then
  setpixel(x, y) with fill_color
  Boundary_fill_4_connected(x,y+1,fill_color, boundary_color)
  Boundary_fill_4_connected(x+1, y, fill_color, boundary_color)
  Boundary_fill_4_connected(x, y-1, fill_color, boundary_color)
  Boundary_fill_4_connected(x-1, y, fill_color, boundary_color)
  end of if
Step 4: End

D. Flood fill algorithm

In some cases want to fill the object when boundary is different, painted directly of interior pixel positions instead go with boundary condition so this flood fill algorithm[16][7] replaces the old color by the fill color when pixels of fill color not exist. Again this algorithm works all adjacent pixel position using four connected and eight connected methods. Below pseudo code is for four connected that can be same for the eight connected then four recursive statements should be added at the end.

Algorithm:

Step 1: Initialize the old_color and fill_color.
Step 2: find the current_value of position (x y).
Step 3: if current_value is equal old_color then
  setpixel(x, y) with fill_color
  Flood_fill_4_connected(x, y+1, old_color, fill_color)
  Flood _fill_4_connected(x+1, y, old_color, fill_color)
  Flood _fill_4_connected(x, y-1, old_color, fill_color)
  Flood _fill_4_connected(x-1, y, old_color, fill_color)
  end of if
Step 4: End

III. COMPARISON AND SIMULATION

Both algorithms have been implemented in TurboC3 platform for four connected as well as eight connected. All the graph based working implemented and plotted in Python which shows the comparison graph between boundary fill and flood fill algorithm. In this paper consider the running time taken by CPU to execute the whole code. These algorithms used recursion process to recall the neighborhood pixels positions, as recursion takes stack as data structure. Table 1 shows the running time (in second) for both algorithms with four connected pixels methods, here flood fill algorithm takes less time than boundary fill algorithm. Figure 4 shows the comparison graph between four connected for both algorithms. In this graph x-axis represents the numbers of pixels and y-axis represents running time of program execution.

The performance of flood fill algorithm is good as comparison of boundary fill algorithm.

Below are the steps to calculate the running time in TurboC3 which has header file "time.h". With the help of clock() function that return the clock_t type value as current time of the system when execution takes place.

Steps for running time calculation in TurboC3 are as follows:

i. Initially include the time header file
ii. Declare the variable of clock_t type
iii. Now put the starting time from where the actual code section starts with clock() function calling
iv. Place the clock() - t statement to find out the running time to execute the code.
v. Finally convert the time into second dividing by CLOCKS_PER_SEC.

Table 1. Running time (in sec) by boundary fill and flood fill algorithm (four connected)

Numbers of Pixels	Boundary fill four connected (Running Time In sec)	Flood fill four connected (Running Time In sec)	Running time difference
625	0.054945	0	0.054945
900	0.054945	0.054945	0
1600	0.164835	0.054945	0.10989
2500	0.21978	0.10989	0.10989
3600	0.32967	0.164835	0.164835

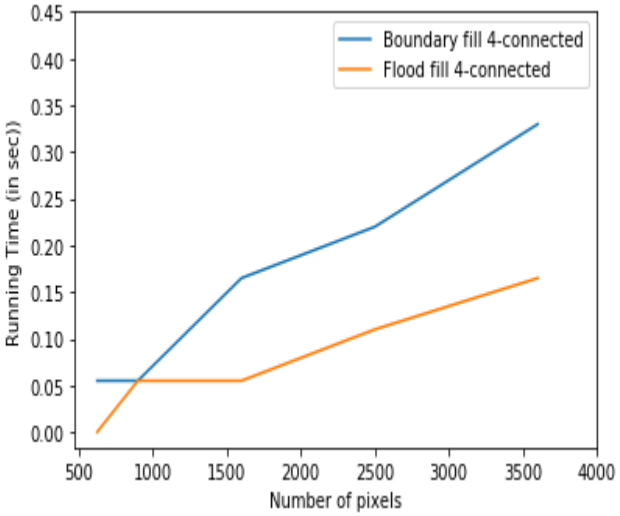

Fig. 4. Comparison between boundary fill and flood fill algorithm four connected

Table 2 shows the running time (in second) for both algorithms with eight connected pixels methods, here flood fill algorithm takes less time than boundary fill algorithm. Figure 5 shows the comparison graph between eight connected for both algorithms. In this graph x-axis represents the numbers of pixels and y-axis represents running time of program execution for both algorithms. The performance of flood fill algorithm is good as comparison of boundary fill algorithm in both case four connected and eight connected.

Table 1 & table 2 both show how boundary fill algorithm takes more running time than flood fill algorithm as the quantity of pixels increase in both case four and eight connected. These tables also has 3rd column which shows the running time difference between algorithms. So we can save the running time when implements the flood fill algorithm. Flood fill saves time of system for large polygon filling. As implement the flood fill algorithm will improve the system performance in case of running time. Running time calculation is measuring in second in c-language simulation.

Table 2: Running time (in sec) by Boundary fill and flood fill algorithm (eight connected)

Numbers of Pixels	Boundary fill eight connected (Running Time In sec)	Flood fill eight connected (Running Time In sec)	Running time difference
625	0.054945	0.054945	0
900	0.164835	0.059495	0.10534
1600	0.274725	0.164835	0.10989
2500	0.43956	0.274725	0.164835
3600	0.549451	0.32967	0.219781

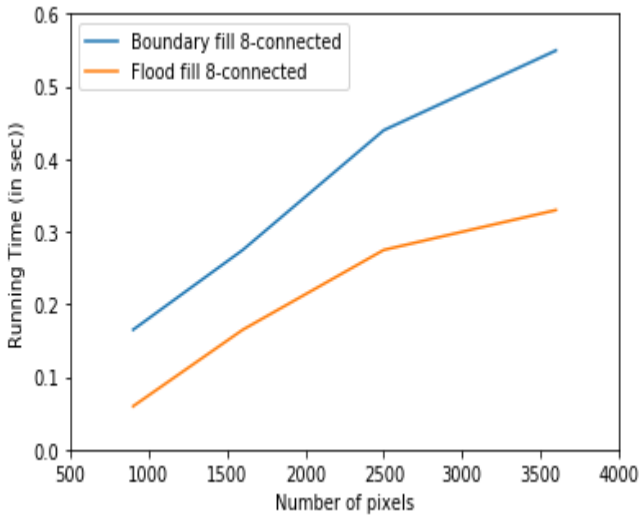

Fig. 5. Comparison between boundary fill and flood fill algorithm 8-connected

In the code section of C-language we consider the polygon is rectangle with multiple sizes like as rectangle (10, 10, 35, 35), rectangle (10, 10, 40, 40), rectangle (10, 10, 50, 50), rectangle (10, 10, 60, 60), rectangle (10, 10, 70, 70). When we fill the above size rectangles then c-language works properly. If we increase the size of rectangle then there is problem to fill the polygon because in both algorithms recursion call will be there which works with stack data structure. So buffer overflow problem will arises over here.

That is reason some part of polygon will be filled and some part still remains same and c-language will not work properly. So that to resolve this problem data structure will be replaced stack by queue. Later on a flood fill algorithm in JAVA with queue data structure works fine with any size of polygon if we increase the height and width of rectangle or increase the number of pixels to fill. That simulation shows the remarkable output of flood fill algorithm with queue i.e how many pixels will be plotted or also calculates the running time (in millisecond) to fill the polygon. We found that if numbers of pixels are 625 then running time is 8 ms, for 1554 pixels takes 10 ms to fill and 3213 pixels takes 13 ms to fill will take and so on shown in table 3.So this simulation of flood fill algorithm of JAVA with queue show the remarkable output. Running time calculation is measuring in millisecond in JAVA simulation. This JAVA simulation finds out the pixels quantity to be filled only.

Below are the steps to calculate the running time in java which has package under "util" package. With the help of nowTime method of System class will return the current time of the system when execution takes place.

Steps for running time calculation in JAVA are as follows:

i. Initially import the package concurrent.TimeUnit under java.util.
ii. Now initialize the long type start_time variable from where the actual code section starts by calling System.nowTime().
iii. Place the long type end_time variable where the actual code section ends by calling System.nowTime().
iv. Calculates running time (in nano second) the difference between start_time and end_time.

Table 3: Running time (in ms) by flood fill algorithm (four connected) using queue

Numbers of Pixels	Flood fill four connected (Running Time in ms)
625	8
1554	10
3213	13
6767	16
9075	17
12604	19

These both simulations work on single processor, if the computer has multi-core of processor or parallel processor then definitely the performance to fill the polygon will improve. In both methods four connected and eight connected, individual processor will take care then the running time will be minimize.

IV. CONCLUSION AND FUTURE WORK

The assessment of complete performance including comparison of boundary fill and flood fill for four and eight connected pixel methods show remarkable results. This work shows the different aspects of parameters for both algorithms. Flood fill algorithm shows the better performance than boundary fill algorithm. In both cases recursion takes place so that output of these algorithms will not work properly due buffer overflow. These algorithms should also be implemented with queue data structure to overcome the buffer overflow problem. With queue, flood fill algorithm work fine for large polygon where number of pixels will be maximized. The open area for further work is the comparison of flood fill and boundary fill with queue data structure. This simulation is done with single processor if same task can also be implemented with multi-processors that will improve the performance to fill the polygon.

REFERENCES

[1] N. Nisha and S. Varshney, "A Review: Polygon Filling Algorithms Using Inside-Outside Test," Int. J. Adv. Eng. Res. Sci., vol. 4, no. 2, pp. 73–75, 2017.
[2] I. Al-rawi, "Implementation of an Efficient Scan-Line Polygon Fill Algorithm," vol. 5, no. 4, pp. 22–29, 2014.
[3] J. Pineda, "A Parallel Algorithm for Polygon Rasterization," vol. 22, no. 4, pp. 17–20, 1988.
[4] E. M. Nosal, "Flood-fill algorithms used for passive acoustic detection and tracking," *New Trends Environ. Monit. Using Passiv. Syst. Passiv. 2008*, vol. 321, pp. 2–6, 2008.
[5] M. R. Dunlavey, "Efficient Polygon-Filling Algorithms for Raster Displays," *ACM Trans. Graph.*, vol. 2, no. 4, pp. 264–273, 1983.
[6] H. Li, "Research and Implementation the Fundamental Algorithms of Computer Graphics Based on VC," no. Meici, pp. 588–593, 2016.
[7] Vipul Aggarwal, "Optimization of Flood Fill Algorithm Using Iterative Look-Ahead and Directional Technique," *Int. J. Comput. Sci. Eng.*, vol. 2, no. 5, pp. 89–94, 2013.
[8] S. Salunke, "Modified Boundary Fill for Complete Surface Coverage by Robotic Agents," *Int. J. Comput. Appl.*, vol. 73, no. 13, pp. 8–11, 2013.
[9] W. G. M. Geraets, A. N. Van Daatselaar, and J. G. C. Verheij, "An efficient filling algorithm for counting regions," *Comput. Methods Programs Biomed.*, vol. 76, no. 1, pp. 1–11, 2004.
[10] H. C. Liu, M. H. Chen, S. Y. Hsu, C. Chien, T. F. Kuo, and Y. F. Wang, "A new polygon based algorithm for filling regions," *Tamkang J. Sci. Eng.*, vol. 2, no. 4, pp. 175–186, 2000.
[11] K. Kallio, "Scanline edge-flag algorithm for antialiasing," *Theory Pract. Comput. Graph. 2007, TPCG 2007 - Eurographics UK Chapter Proc. Celebr. 25 Years Eurographics UK Chapter*, pp. 81–88, 2007.
[12] D. Henrich, "Space-efficient Region Filling in Raster Graphics," vol. 10, no. 4, pp. 205–215, 1994.
[13] J. Arvo, M. Hirvikorpi, and J. Tyystjärvi, "Approximate soft shadows with an image-space flood-fill algorithm," *Comput. Graph. Forum*, vol. 23, no. 3 SPEC. ISS., pp. 271–279, 2004.
[14] K. Muthukumar, S. Poorani, and S. Sindhu, "Color Image segmentation using Similarity based Region merging and Flood Fill Algorithm," vol. 5, no. 06, pp. 40–46, 2016.
[15] A. Javeed, "a Novel Region Filling Algorithm for Discontinuous Contours," *Int. J. Res. Eng. Technol.*, vol. 06, no. 12, pp. 18–23, 2017.
[16] C. Bond, "An Efficient and Versatile Flood Fill Algorithm for Raster Scan Displays," 2011.

AUTHORS PROFILE

Bhawnesh Kumar, Lecturer, Department of Computer Science, D. A. V. College Muzaffarnagar (U.P), India. MCA from Kurukshetra University, Kurukshetra, Haryana, India, M.Tech (CSE) from R. G. T. U Bhopal, M.P (State University), India. He has published five research papers. His areas of interest are clustering in wireless sensor network, computer graphics.

Dr. Umesh Kumar Tiwari is working as an Associate Professor in Department of Computer Science and Engineering in Graphic Era Deemed to be University, Dehradun. He had received his Ph.D. in 2016. He has more than 12 years of experience in teaching/research of UG and PG level degree courses as a Lecturer/Assistant Professor/ Associate Professor in various academic/research organizations. He is supervisor of 02 PhD and 5 M. tech students who are working on specific domains of software engineering and network security. His research is on multidisciplinary topics and he has published 17 journal papers in reputable international and national journals, and 12 conference papers in reputed international and national conferences. His research interests are Wireless Communication Networks, Network Security, and Software Engineering topics with improved modeling, interaction-integration complexities, testing and reliability models.

Dr. Santosh Kumar had received his Ph.D. from IIT Roorkee (India) in 2012, M. Tech. (CSE) from Aligarh Muslim University, Aligarh (India)in 2007 and B.E. (IT) from C.C.S. University, Meerut (India) in 2003. He has more than 13 years of experience in teaching/research of UG (B. Tech.) and PG (M.Tech.) level courses as a Lecturer/Assistant Professor/ Associate Professor in various academic /research organizations. He has supervised 01 Ph.D. Thesis, 20 M.Tech. Thesis, 18 B.Tech projects and presently mentoring 06 Ph.D students, 03 M.Tech students and 04 B.Tech. students. He has also completed a consultancy project titled “MANET Architecture Design for Tactical Radios” of DRDO, Dehradun in between 2009-2011.He is an active reviewer board member in various national/International Journals and Conferences. He has memberships of ACM (Senior Member), IEEE, IAENG, ACEEE, ISOC (USA) and contributed more than 46 research papers in National and International Journals/conferences in the field of Wireless Communication Networks, Mobile Computing and Grid Computing and software Engineering. Currently holding position of Associate professor in the Graphic Era Deemed to be University, Dehradun (India). His research interest includes Wireless Networks, MANET, WSN, IoT, and Software Engineering.

Vikas Tomer Assistant Professor, Department of Computer Science and Engineering, Graphic Era Deemed to be University, Dehradun, India. M.S (Data Analytiics) from National College of Ireland, Dublin, Ireland, M.E (Computer Science) from Punjab Engineering College, Chandigarh, India, B.Tech (IT) From U.P.T.U, Lucknow, India. His research area is data science and machine learning.