*International Journal of Educational Technology and Artificial Intelligence*

ISSN 2964-5387 (online)
Vol 3, No 1, 2024

# Visualizing the Digital Differential Analyzer (DDA) algorithm using Adobe Animate

**Monica Troka**
University of Almeria, Spain, moniktroka@gmail.com

**Abstract**

This research presents an innovative approach to visualizing the Digital Differential Analyzer (DDA) algorithm using Adobe Animate, a versatile tool known for its animation and graphics capabilities. The Digital Differential Analyzer is a fundamental algorithm in computer graphics, and our work focuses on creating an engaging and educational visualization that demystifies its step-by-step execution.The project begins by designing a comprehensive storyboard that outlines the key components and stages of the DDA algorithm. Leveraging Adobe Animate's user-friendly interface, we implement frame-by-frame animations that dynamically illustrate the algorithm's progression. Each frame depicts a specific step in the DDA process, with graphical elements representing points, lines, and the calculated variables involved in drawing a line.The visualization incorporates interactive elements, allowing users to control parameters such as slope and starting coordinates. By utilizing Adobe Animate's scripting capabilities, we enable real-time updates and dynamic feedback, enhancing user engagement and understanding. The inclusion of sliders, buttons, and dynamic text labels empowers users to actively manipulate and observe the algorithm in action.

**Keywords:** DDA algorithm, graphic primitive, image, visualization

## I. Introduction

Images can be described in several ways. When using a raster-display, the image is defined by a set of intensities for pixel positions on the display [1].. Images are complete objects, such as trees, furniture, and others that are placed at certain coordinates in the scene. The shape and color of an object can be defined by an array of pixels or a set of basic geometric structures, such as straight lines and colored areas of polygons.

Scenes display images by loading an array of pixels into the frame buffer or by converting a scan of a specified graphic geometry into a pixel pattern. Graphics programming packages are equipped with functions to express scenes in the form of basic geometric structures called output primitives by inputting these primitive outputs as more more complex structures [2]. Each output primitive has coordinate data and other information about how the object is displayed on the screen. Points and straight lines are the simplest geometric shapes of image components. Promisive outputs that can be used to form images include circles, spline curves, conics, and others.

We will first discuss the image formation procedure by examining an algorithm that displays output primitives in two dimensions. In this chapter we also study how primitive output functions are used in graphics application packages.

### A. Points and Lines

Point formation is carried out by converting a coordinate position with an application program into a certain operation using output equipment. With a CRT monitor, a beam of electrons appears and disappears to light the phosphor. How the electron beam determines the position of the point depends on the display technology used.

Received 11 September 2023, Revised 28 Octobrer 2023, Accepted 28 October 2023, Available online 5 February 2024, Version of Record 28 October 2023.

ISSN 2964-5387 (online)
Vol 3, No 1, 2024

The random-scan (vector) system stores point formation instructions in the display list and the coordinate values determine the position of the electron beam towards the phosphor layer on the screen. For black and white raster scans, a point is determined by assigning a value of 1 to a certain position on the screen. In the RGB system, the intensity color code for pixel positions on the screen is stored in the frame buffer.

Lines are created by determining the position of points between the start and end points of a line [3]. Then, the output equipment creates lines according to the positions of these points. For analog equipment, such as plotters and random-scan displays, straight lines can be produced smoothly. Meanwhile, on digital equipment, straight lines are produced by setting discrete points between the start and end points. The position of a discrete point along a straight line can be calculated from the equation of that line. Point calculations that produce fractional values, such as (5.37, 10.78) are converted to pixel positions (5, 11). Rounding the coordinate values to an integer results in a line displayed on the screen resembling a ladder image, as in Figure 1. The smoothness of the display on the screen will depend on the resolution used.

*Figure 1 A line that is genaerated by pixels*

Pixel positions can be described according to scan-line values and column values (pixel positions made perpendicular to the scan-line). The scan-line value starts with 0 at the bottom of the screen, and the pixel column starts with 0 from the left of the screen. To determine the value of a point, a basic procedure can be used where x is the pixel column value and y is the scan-line value, as follows:

setPixel (x, y)

If the x and y values have been stored in the frame buffer, you can display them on the screen using basic functions

getPixel (x, y)

## B. Line Formation Algorithm

The equation of the line according to Cartesian coordinates is

$y = m . x + b$

where m is the slope of the line formed from two points, namely (x1, y1) and (x2, y2) as can be seen in Figure 3.3. To add x along the line, namely dx, you will get an increase in y of

$dy = m .dx$

In a raster system, lines are formed based on pixels, with step sizes in the horizontal and vertical directions. Thus, the position of each pixel must be determined by a discrete position value or something close to it.

### *DDA Algorithm*

Digital Differential Analyzer (DDA) is a line formation algorithm based on dx and dy calculations, using the formula dy = m.dx. Lines are created by determining two endpoints, namely the start point and the end point. Each coordinate of the point that forms the line is obtained from calculations, then converted into an integer value.

The steps to form a line according to the DDA algorithm are as follows:

1. Determine the two points that will be connected to form a line.
2. Determine one of the points as the starting point (x0, y0) and end point (x1, y1).
3. Calculate dx = x1- x0, and dy = y1- y0.
4. Determine the step, namely the maximum distance between adding x and y values, by:
   - If the absolute value of dx is greater than the absolute value of dy, then step = absolute of dx
   - If not, then step = absolute of dy
5. Calculate the additional pixel coordinates, namely x_increment = dx/step, and y_increment = dy/step.
6. Next coordinates (x + x_increment, y + y_increment)
7. The pixel position on the screen is determined by rounding the coordinate values.
8. Repeat numbers 6 and 7 to determine the next pixel position, until x = x1 and y = y1.

ISSN 2964-5387 (online)
Vol 3, No 1, 2024

## II. Method

Creating a visualization of the Digital Differential Analyzer (DDA) algorithm using Adobe Animate [4] involves leveraging the software's animation and graphics capabilities. Adobe Animate is well-suited for this task, providing a user-friendly interface for designing interactive and visually appealing content. Here's a step-by-step exploration of how you could approach visualizing the DDA algorithm using Adobe Animate:

*Step 1: Storyboard Design*

Before diving into Adobe Animate, plan a storyboard [5] that outlines the key steps and visual elements of the DDA algorithm. Consider how each frame or scene will depict the progression of the algorithm, from initializing variables to drawing the line.

*Step 2: Graphics and Animation*

*Canvas Setup:*
- Open Adobe Animate and create a new project. Set up your canvas size and background.

*Basic Line Drawing*:
- Use Adobe Animate's drawing tools to create a simple coordinate system. Design a point or small line to represent the initial pixel on the screen.

*Frame-by-Frame Animation:*
- Utilize the timeline to create frames that represent each step of the DDA algorithm.
- Gradually move or duplicate elements to simulate the progression of drawing the line.

*Variables and Labels:*
- Integrate text labels and dynamic fields to display the values of variables involved in the algorithm (e.g., x, y, slope).

*Step 3: Interaction (Optional)*

*User Interaction:*
- Implement interactivity to allow users to control aspects of the DDA algorithm. This could include sliders for adjusting line parameters or buttons to step through the algorithm.

*Step 4: Advanced Visuals (Optional)*

*Color and Styling:*
- Enhance visual appeal by incorporating color gradients, different line styles, or shading.

*Step 5: Testing and Refinement*

*User Feedback:*
- Gather feedback from users or colleagues to identify areas for improvement.

By following these steps, you can leverage Adobe Animate to create an engaging and informative visualization of the Digital Differential Analyzer algorithm. This approach combines animation, interactivity, and advanced graphics to enhance the learning experience for users interested in understanding the intricacies of the DDA algorithm.

## III. Results and Discussion

To illustrate how to develop multimedia application that contains algorithm visualization in Adobe Animate, the first step is to define the learning objectives [7]. The tutorial involves many topics such as graphic systems, input and output primitives, and graphic programming or scripting.

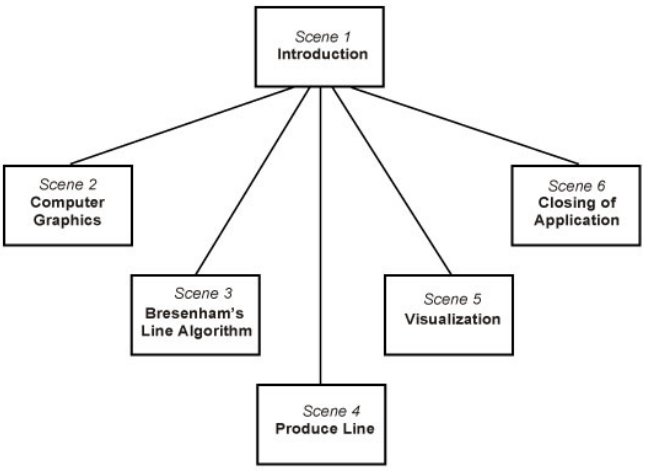


*Figure 3. Navigation structure*

The next stepis establishes lateral thought processes, helping to break down the navigation structures that are usually embedded in traditional approaches to course delivery. Then, it can result in an overview based on quite abstract design, which in turn generates

ISSN 2964-5387 (online)
Vol 3, No 1, 2024

fresh implementation. Finally, it provides a storyboard for identifying relationships between the components. Navigation structure is essential to design an interactive multimedia application shown in Figure 3.

## A. Scripting

The script of a button for calculating to discover pixels, as follow:

```
on(release) {
        dx=number((x2)-(x1));
        dy=number((y2)-(y1));

        if(Math.abs(dx)>Math.abs(dy)){
                        steps=Math.abs(dx);}
                else if(Math.abs(dx)<Math.abs(dy)){
                        steps=Math.abs(dy);}
                else {
                        steps=Math.abs(dx);}

                xplus=dx/(steps);
                yplus=dy/(steps);
                x=x1;
                y=y1;
                myArray=new Array();

                for(var i=0;i<steps;i++){
                        xa=1.0*x+xplus;
                        x=1.0*x+xplus;
                        myArray[i]=Math.round(xa);
                }
                outputx=myArray;
                myArray=new Array();
                for (var j=0;j<steps;j++){
                        ya=1.0*y+yplus;
                        y=1.0*y+yplus;
                        myArray[j]=Math.round(ya);
                }
                outputy=myArray;
                output=»x      «+x1+»   «+outputx.join(«
«)+newline+»y  «+y1+» «+outputy.join(« «);
   }
```

The visualization is ganerated, as shown in Figure 4. The script of a button for presenting visualization, as follow:

```
on (release) {
        a=x1*10+100;
        b=y1*(-10)+400;
        c=x2*10+100;
        d=y2*(-10)+400;
        _root.createEmptyMovieClip(«garis»,1);
        _root.garis.lineStyle(2);
        _root.garis.moveTo(a,b);
        _root.garis.lineTo(c,d);
}
```

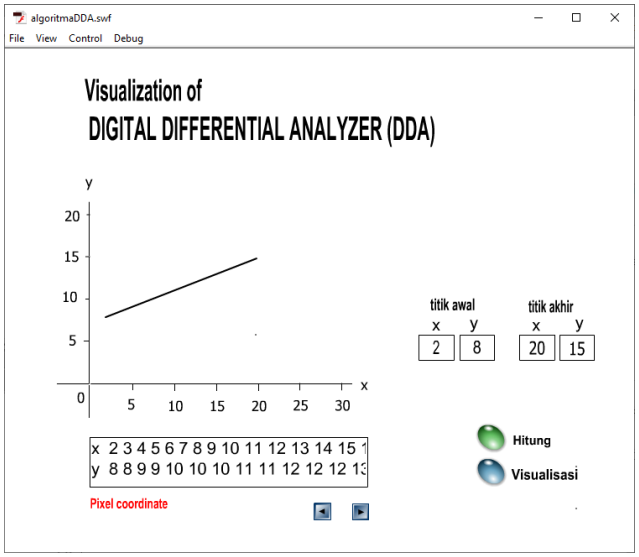


*Figure 4. DDA algorithm visualization*

## B. Limitations and issues

Some limitations and issues, which are:

- Floating point arithmetic: The DDA algorithm requires floating-point arithmetic, which can be slow on some systems. This can be a problem when dealing with large datasets.
- Limited precision: The use of floating-point arithmetic can lead to limited precision in some cases, especially when the slope of the line is very steep or shallow.
- Round-off errors: Round-off errors can occur during calculations, which can lead to inaccuracies in the generated line. This is particularly true when the slope of the line is close to 1.
- Inability to handle vertical lines: The DDA algorithm is unable to handle vertical lines, as the slope becomes undefined.
- Slow for complex curves: The DDA algorithm is not suitable for generating complex curves such as circles and ellipses, as it requires a large number of line segments to approximate these curves accurately.
- Aliasing: Aliasing occurs when the line segments generated using the DDA algorithm do not accurately represent the line being

ISSN 2964-5387 (online)
Vol 3, No 1, 2024

drawn, resulting in a jagged appearance.
- Not suitable for thick lines: The DDA algorithm generates thin lines, which can be problematic when drawing thick lines, as the line segments may overlap or leave gaps.

## IV. Conclusion

Through this paper the DDA Algorithm visualization has been presented. Some details about graphic programming that should be learned by students have been described. The visualization and the interactivity have been well tested by the students at auniversity. Adobe Animate is a timeline-based, authoring and object-oriented programming tools can be used to develop a scientific visualization..

## V. Acknowledgement

The authors thank all the survey respondents and participants willing to participate in this research project consciously and voluntarily.

## References

[1] D. Hearn and P. Baker, Computer Graphics. Englewood Cliffs: Prentice Hall International, Inc., 2006.

[2] H. Sutopo, "Bresenham's Lines Algorithm Visualization Using Flash," Int. J. Comput. Theory Eng., vol. 3, no. 3, pp. 422–426, 2011, doi: 10.7763/ijcte.2011.v3.342.

[3] F. S. Hill Jr, Computer Graphics Using Open GL. Upper Saddle River, NJ: Prentice Hall International, Inc., 2001.

[4] R. Chun, Adobe Animate CC Classroom in a Book. California: Adobe Press, 2019.

[5] F. N. Kumala, A. Ghufron, P. P. Astuti, M. Crismonika, M. N. Hudha, and C. I. R. Nita, "MDLC model for developing multimedia e-learning on energy concept for primary school students," J. Phys. Conf. Ser., vol. 1869, no. 1, 2021, doi: 10.1088/1742-6596/1869/1/012068.