frontiers | Frontiers in Computer Science

TYPE Perspective
PUBLISHED 03 July 2025
DOI 10.3389/fcomp.2025.1520930

OPEN ACCESS

EDITED BY
Kostas Karpouzis,
Panteion University, Greece

REVIEWED BY
Ruben Cornelius Siagian,
State University of Medan, Indonesia

*CORRESPONDENCE
Luiz Anastácio Alves
alveslaa40@gmail.com

RECEIVED 31 October 2024
ACCEPTED 10 June 2025
PUBLISHED 03 July 2025

CITATION
Oliveira R, Dias EA, Santos R, Cotta-De-Almeida V, Aguiar Coelho Nt J and Alves LA (2025) How math shapes the world of life science animation.
Front. Comput. Sci. 7:1520930.
doi: 10.3389/fcomp.2025.1520930

How math shapes the world of life science animation

Rafael Oliveira[1], Evellyn Araujo Dias[1,2], Ricardo Santos[1,3], Vinicius Cotta-De-Almeida[4], José Aguiar Coelho Nt[1,5,6] and Luiz Anastácio Alves[1]*

[1]Laboratory of Cellular Communication, Oswaldo Cruz Institute, Oswaldo Cruz Foundation, Rio de Janeiro, Brazil, [2]Laboratory of Innovations in Therapies, Education and Bioproducts; Oswaldo Cruz Institute, Oswaldo Cruz Foundation, Rio de Janeiro, Brazil, [3]Faculty of Technological Education of the State of Rio de Janeiro (FAETERJ), Rio de Janeiro, Brazil, [4]Laboratory on Thymus Research, Oswaldo Cruz Institute, Oswaldo Cruz Foundation, Rio de Janeiro, Brazil, [5]National Institute of Industrial Property - INPI, Patent Examination Division XV, Rio de Janeiro, Brazil, [6]Veiga de Almeida University, Electric Engineering Faculty - Campus Tijuca, Rio de Janeiro, Brazil

With the advances in technology, it is noticeable the educational potential of animation in the field of cell biology, physiology and other basic life science disciplines, revolutionizing the learning process in science. This paper elucidates the role of matrix manipulation in animating figures on screens, elucidates the distinctions between Scalable Vector Graphics (SVG), bitmap and raster images, and unveils the inner workings of the Bresenham algorithm in the context of rendering lines on screens. Furthermore, the article offers a practical dimension to this theoretical understanding by providing a comprehensive example of code written in JavaScript for generating an animation, also using HTML and CSS. This code example is designed to be easy to comprehend, even to those with limited programming experience, fostering the integration of animations into life science education. We synthesized findings from various studies to underscore the proven advantages of this new teaching tool, and by reviewing them, we reaffirm that animations have a demonstrable impact on improving the learning experience, making complex physiological processes more comprehensible and engaging. This highlights the critical role of animations as a pedagogical tool in science classrooms, and emphasizes the importance of understanding the mathematical and computational principles that support their creation. By bridging this knowledge gap, educators and students can make more effective and informed use of existing animation tools.

KEYWORDS

animation, biology, math, secondary and undergraduate, programming

1 Introduction

Fields like cell biology, physiology, and biochemistry are crucial for understanding disease mechanisms. However, their study often leans heavily on memorization, making it tedious. Textbooks, being the primary information source, may fail to offer clear, engaging explanations (Kalas and Redfield, 2022). To overcome this, animations in these textbooks can aid in illustrating complex concepts and phenomena. Animations not only capture visual attention, enhancing long-term memory, but also simplify understanding of dynamic biological processes (O'Day, 2007; Wildhaber et al., 2011; Peart et al., 2022). They effectively demonstrate time and space-related changes like protein movements, enzymatic reactions, phagocytosis processes, cell division. Animations attract and maintain attention, proving beneficial in learning, marketing, and teaching (O'Day, 2007; Praveen and Srinivasan, 2022). Praveen et al.'s

review (2015–2021) on animations' impact on attention highlights their role in enhancing learning strategies.

In this digital age, the realm of teaching is being revolutionized with technology, turning classrooms into dynamic learning environments. This approach is very effective in demystifying the complexities of biology. Through virtual tools, what was once abstract becomes visually tangible, enhancing student understanding (Veselinovska and Stavreva, 2020). A popular and powerful tool to help better understand biological processes is animation. Table 1 shows a few softwares available for the purpose of animation and illustration. With platforms like YouTube, Instagram, and user-friendly software such as Inkscape, creating and sharing animations has never been easier (Coluci, 2022). This fusion of technology and pedagogy is reshaping how we teach and learn.

However, while these tools have become more accessible, there is still a gap in understanding the fundamental principles behind how they work. This paper does not suggest that animation via code is inherently better than using dedicated software. Rather, we argue that by understanding the mathematical and computational foundations that power these tools—such as matrix operations, vector geometry, and rendering algorithms—educators and students can use animation software more effectively and critically. This knowledge enhances their ability to customize animations, ensure scientific accuracy, and develop a more meaningful interpretation of the biological processes being represented.

While technology has advanced the use of animations in education, there is still a gap in understanding the core process of creating these animations. It is crucial to grasp what goes on behind the software's interface to better understand how it works and how to use it properly to create more realistic biological animations. This article explains some important concepts used in the field of computer graphics and the mathematics behind it, to illustrate how the process of creating an animation works. This includes knowledge about how matrices are utilized in this context and the algorithms involved in generating vector figures. Understanding these foundational elements is not necessarily about replacing animation tools, but about enhancing their use. By grasping the mathematics and logic behind them, users can create more accurate and effective biological animations and better evaluate the visual models these tools generate.

TABLE 1 Free softwares for illustration and animation.

Softwares	Purpose	Website
Krita	Illustration and animation	https://krita.org/
GIMP	Illustration	https://www.gimp.org/
Inkscape	Illustration	https://inkscape.org/
FireAlpaca	Illustration	https://firealpaca.com/
MediBang Paint	Illustration	https://medibangpaint.com/
Blender	Illustration and animation	https://www.blender.org/
Pencil2D	Illustration and animation	https://www.pencil2d.org/
Synfig Studio	Illustration and animation	https://www.synfig.org/
Vectorian Giotto	Illustration and animation	https://archive.org/details/vectoriangiotto_201807
Google Web Designer	Illustration and animation	https://webdesigner.withgoogle.com/intl/pt-BR/
Plastic Animation Paper (PAP)	Animation	https://www.plasticanimationpaper.dk/
Opentoonz	Animation	https://opentoonz.github.io/e/

2 Vectors and matrices

Mathematics plays a pivotal role in creating accurate and informative animations of biological processes. For example, when showing how a cell divides, mathematical algorithms can control the movement of chromosomes and the shape of the cell as it splits. In animations of blood circulation, curves and equations are used to draw smooth, realistic paths that represent how blood flows through veins and arteries. These mathematical tools allow animations to show both structure and movement in ways that are easier to understand, helping students visualize processes that are often hard to grasp through static images or text alone.

Matrices and vectors are mathematical concepts with wide-ranging applications, such as in computer graphics. Understanding matrices and vertices is fundamental to mastering animation and computer graphics, as they form the backbone of how visual data is represented and manipulated within digital environments. This article explains their use in manipulating space through matrix operations and transformations in computer graphics. It covers the basics of matrices and vectors, their role in representing space, and how they are represented in programming languages.

Vectors are essentially ordered lists of elements, useful in representing directions in space. In computer graphics, they are used for positions and directions, with 2D coordinates represented by two indexes (x and y) as represented in Supplementary Figure 1A, and 3D or 1D space represented by adding or removing components.

In computer graphics, a matrix is a rectangular array of numbers, symbols, or expressions arranged in rows and columns. Two representations of matrices and their indexes are shown in supplementary Figure 1B, the first one representing its indexes as in mathematics, and the second as in the JavaScript programming language. Matrices are used to perform various mathematical operations like shearing, rotations, translations, and scaling, which are crucial for modeling and rendering images in both 2D and 3D spaces. Essentially, a vector can be seen as a one-dimensional matrix, with its indexes representing coordinates. Shapes in space can be represented by a collection of vectors that define the coordinates of their vertices, organized within a matrix, where each column corresponds to a vector representing a vertex. The connections between these vertices form the structure of the shape.

3 Matrix transformations for shapes

To transform shapes in computer graphics, various matrix transformations are used. These include translation (moving objects without changing their form), rotation (changing the object's angle), scaling (altering the object's size), and shearing (modifying the shape and size along axes). Additionally, matrix addition can combine

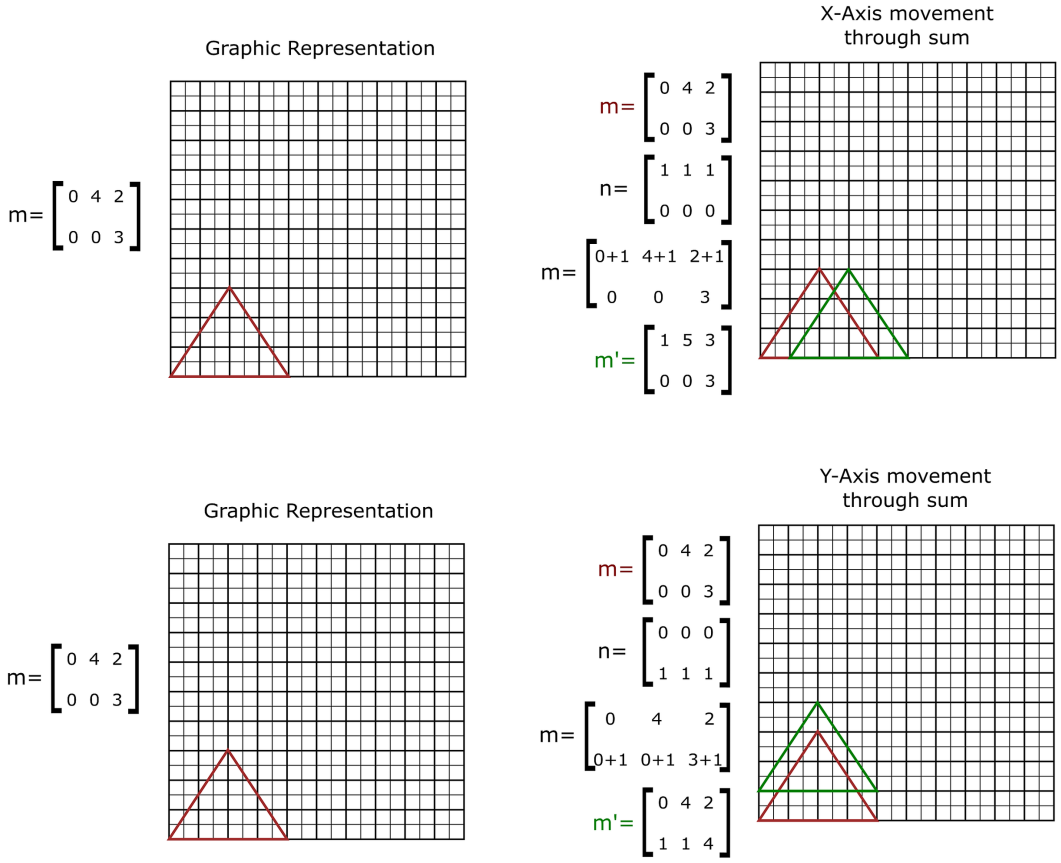

FIGURE 1
Representation of the movement of a triangle through the x and y-axis by addition of matrices m and n, which have 3 vectors that represents coordinates, resulting in a new matrix m'.

matrices for complex transformations, useful in creating movement in animations as in Figure 1, which illustrates horizontal and vertical movement, and in Supplementary Figure 2, which illustrates diagonal movement. Scalar multiplication changes the length of a vector without altering its direction as shown in Supplementary Figure 3, essential for scaling objects uniformly and making vector art resolution-independent. These transformations play a crucial role in animating and manipulating objects in computer graphics (Kist, 2020).

In game development platforms, translation, rotation, and scaling matrices are fundamental for animating characters, controlling object movement, and simulating realistic environments. These transformations are often applied iteratively, frame by frame, to produce smooth and continuous animations. In 3D modeling software like Blender, matrix operations are essential for transforming meshes, adjusting camera perspectives, and applying lighting effects during both modeling and animation workflows. Similarly, in OpenGL—one of the most commonly used graphics libraries—transformation matrices are part of the rendering pipeline, enabling efficient manipulation of objects in real time and supporting animation through timed updates of matrix values. These practical examples demonstrate how matrix mathematics underpins many of the visual effects and animated representations commonly used in scientific visualization, biology education, and entertainment media.

Besides helping to create movement and transformations, matrices also play an important role in how fast and smoothly animations run—especially in real-time environments like simulations or educational apps. When many objects are moving or changing at once, the computer needs to do a lot of calculations quickly. Matrix operations are efficient because they follow simple math rules that computers can process very fast. In most animation systems, different transformations (like moving, rotating, and resizing an object) can be combined into a single matrix, reducing the number of calculations needed. This makes animations smoother and helps them run well even on basic computers or in classroom settings. By using optimized matrix calculations, it's possible to show complex biological processes, like the beating of a heart or cell division without lag.

To bring these shapes to life on a digital screen, we must also understand how they are rendered using pixels and color. Therefore, it is important to understand how digital screens interpret spatial information through coordinate systems and how RGB color models are used to represent visual details in a way that supports both accuracy and clarity in animated content.

4 Screen and RGB

Understanding how images are displayed on screens involves different concepts compared to a Cartesian grid. While screens share similarities with Cartesian grids—such as having specific coordinates for pixel locations—there are key differences. One

notable distinction is that, on screens, the y-axis is inverted compared to the Cartesian grid, as illustrated in Supplementary Figure 4. Each coordinate on the screen represents a pixel, which typically consists of three subpixels colored Red, Green, and Blue. This model combines different intensities of these three colors to produce a wide range of colors (Zelazko, 2023). Unlike the Cartesian grid, screens only use integer coordinates and have a fixed resolution, determining the sharpness and detail of the image. In digital color representation, each color's intensity is typically stored in a three-index vector, like {255, 255, 255}. Each index in this vector represents a primary color (Red, Green, Blue) and can have a value ranging from 0 to 255. By adjusting these values and mixing different intensities, a wide range of colors can be created. For instance, a vector of {255, 255, 255} would represent the color white. This system allows for the precise representation of colors on digital screens. Some examples of visualization can be seen in Supplementary Figure 5, which shows 3 vectors with 3 indexes each, and the color that they represent in the RGB model.

While RGB is the standard color model for digital displays, it is not the only method for representing color. Other models, such as CMYK (Cyan, Magenta, Yellow, Key/Black) and HSL (Hue, Saturation, Lightness), offer different approaches to color representation. CMYK is a subtractive color model commonly used for production printers, where colors are created by subtracting varying amounts of ink from a white background. In contrast, HSL is a cylindrical-coordinate representation of points in an RGB color model, designed to be more intuitive and perceptually relevant than the Cartesian (cube) representation. It defines colors in terms of three components: hue (the type of color), saturation (the intensity of the color), and lightness (the brightness of the color). Understanding these alternative models is crucial, as the RGB system has limitations, especially when colors are reproduced on different media. A color that appears vibrant on a screen may look dull when printed, due to the differences in how RGB and CMYK models represent colors (Ibraheem et al., 2012; Nayyer and Sharma, 2015; Shishmanova and Rinaldi, 2018).

Screen resolution refers to the number of pixels arranged horizontally and vertically on a digital display, and it directly influences the clarity and detail of an image. For example, a 720p screen has a resolution of 1,280 × 720 pixels, while a 4 K screen displays at 3,840 × 2,160 pixels—offering significantly more visual information. This difference becomes especially important in educational animations, where fine structures such as organelles or cell membranes must be represented clearly. On a lower-resolution screen, these details may appear blurry or pixelated, reducing the effectiveness of the visualization. In contrast, higher-resolution displays allow animations to present smoother lines, sharper edges, and more precise colors, resulting in a more accurate and engaging learning experience.

Behind every image rendered on a screen lies a particular type of graphic structure—either raster-based or vector-based. These two formats, bitmap and SVG, differ not only in how they store and organize visual information but also in how they behave when scaled, manipulated, or animated (Eisenberg, 2002). To deepen this understanding, the following section will explore the characteristics, advantages, and limitations of these image formats, and how each one plays a role in the construction of animations and illustrations.

5 SVG and bitmap

Imagine a digital world where matrices, vectors, and vertices are magical tools for creating and transforming images. In this world, there are two main types of images: raster and vector. Supplementary Figure 6A shows that a raster image works like digital pointillism, where tiny colored dots come together to form a picture (Bosch and Heman, 2005). Zoom in as illustrated in Supplementary Figure 6B, and you'll see each individual pixel, much like looking closely at a pointillist painting.

On the other hand, vector images are like detailed blueprints describing shapes and lines. These blueprints adjust themselves to maintain perfection, no matter how much you resize or transform them. It's like having a magic wand that keeps everything sharp and clear. A great example is SVG, a format that stores all the instructions for creating these magical vector images. Supplementary Figure 7A illustrates that an SVG is essentially a text file, which in this instance establishes an area of specific width and height. Within this area, it defines a line connecting two points with specified coordinates, colored according to the stroke property. While this example features a simple line, SVG files can describe much more complex images. SVGs have the advantage of being able to render geometrical shapes after transformations, allowing shape manipulation without quality loss, as shown in Supplementary Figure 7B. However, vector illustrations generally require less storage space compared to raster images, as they describe shapes instead of just color points. It is also important to note that on a pixel-based monitor, vector art is converted to raster format for rendering, maintaining the perception of unchanged quality during transformations (Eisenberg, 2002).

Regarding storage, raster files, especially high-resolution ones, tend to be larger due to the need to store color information for each pixel. This can impact loading times and storage space, particularly on web platforms. Vector files, however, generally have smaller sizes since they store only the mathematical descriptions of shapes and colors, making them more efficient for web use.

Understanding these distinctions is crucial when rendering vector graphics onto raster displays, a process that involves converting mathematical descriptions into pixel-based representations. This conversion is where algorithms like Bresenham's line algorithm become essential, as they efficiently determine which pixels best approximate the intended vector shapes, ensuring accurate and visually appealing results on screen.

6 Bresenham algorithm

Now knowing the basics of how a screen works, what is a vectorial image and the differences between a raster image, we can dive on the fundamentals of how vectorial shapes can be rendered. For that we must understand how programs and operating systems render information and the algorithm which converts a vectorial line to its raster image representation.

The text content of Supplementary Figure 7A is a description of the drawing of Supplementary Figure 7B, with a width and height of 100, which has a line inside which goes from the coordinates defined on the points x1, y1 to x2, y2. It also has a property stroke, which has the value black, that symbolizes the color the line should assume. As discussed before, the SVG file only gives us information about what should

be rendered in the figure, so it does not render it on the screen. That would be the job of a rendering engine which could be used by a browser, an editing software or even a library used by the operating system. In this case, we will discuss xorg-server and how it renders a line.

X. Org (commonly referred to as Xorg) is the default X Window System server in Unix-like operating systems, widely used in both Linux and BSD. This server is open source and widely used, and its primary function is to provide a platform for displaying graphical elements on a computer screen and managing input devices like keyboards and mouses. It acts as an intermediary between applications (such as browses, drawing software or image visualizers) and the display hardware. In the xorg-server source code, there is an implementation of the Bresenham algorithm for computer control of a digital plotter, which can be used for example to render a straight vectorial line (such as the one described in the SVG file) as a raster figure, making it possible to render it on the screen (Bresenham, 1965). Xorg provides an implementation of the Bresenham algorithm available on the Xorg server GitHub repository. Instead of delving into Xorg's specific implementation, we will use a basic JavaScript function that operates on the same principle. If you are interested in exploring how Xorg implements Bresenham's algorithm, their source code is available online.[1]

When teaching the principles of computer graphics, especially line drawing, the Bresenham line drawing algorithm serves as an excellent example to illustrate how complex visual outcomes can be achieved through simple mathematical logic. This algorithm provides a foundational lesson in both the power of algorithmic thinking and the practical application of mathematical concepts in computer science.

To understand the Bresenham algorithm, let us start with the basics. Imagine you are tasked with drawing a straight line between two points on a piece of graph paper, but all you have is a pencil, and you aim to draw this line as straight as possible without the aid of a ruler. This scenario closely mimics the challenge faced in computer graphics when attempting to render a straight line between two points on a pixelated display.

In computer graphics, the screen can be thought of as a grid of pixels, similar to the squares on graph paper. The challenge is determining which pixels to "light up" so that they collectively form what appears to be a straight line to the human eye. This is where the Bresenham algorithm comes into play, acting as a guide to select the most appropriate pixels for this task.

The beauty of the Bresenham algorithm lies in its simplicity and efficiency. It uses only integer addition and subtraction to make decisions, avoiding the computational overhead associated with floating-point arithmetic. This simplicity is critical for real-time rendering, where computational speed is paramount, such as in video games or interactive applications.

Here's how the algorithm works in a pedagogical context: Starting Point: The algorithm begins at the first pixel or point; Decision Making: At each step, it evaluates the pixels that lie along the path of the line and selects the next pixel that will keep the line as straight as possible, based on the line equation $y = mx + c$, where m is the slope of the line and c is the y-intercept; Adjustment for Steep Lines: If the line is particularly steep, the algorithm adjusts its selection strategy to ensure the visual integrity of the line remains intact; Practical Application: Imagine drawing a line from two coordinates such as (0, 0) to (15, 10). Direct drawing might lead to discrepancies between the actual and desired line paths due to pixel alignment issues. The algorithm solves this by determining which pixel best represents each segment of the line, based on their proximity to the theoretical line path; Calculation of Distances: This involves computing the distances between the theoretical line and the candidate pixels, using these calculations to make informed decisions about which pixel to illuminate; Handling Cases of inclination: When a line has a high slope, greater than 1, it can be challenging to draw it accurately using the basic Bresenham algorithm. In such cases, the algorithm adjusts its strategy to ensure the line's visual integrity remains intact. For lines with a slope greater than 1, the algorithm swaps the roles of x and y coordinates. This effectively treats the line as if it were sloping less than 1, simplifying the rendering process. In summary, the Bresenham algorithm is a fundamental tool in computer graphics, providing a simple yet effective method for rendering straight lines on a pixelated display. By understanding its principles and adaptations for different line slopes, developers can achieve efficient and accurate line rendering in various applications.

In the context of Figure 2A, Bresenham's algorithm tackles a fundamental challenge in rasterization: determining which theoretical point, M1 or M2, should be rendered on the screen based on the distance to M. Figure 2B elucidates that the algorithm operates by evaluating the distances d1 between M1 and M and d2 between M2 and M. The initial error, denoted as P1, crucially arises from the combination of d1 and d2, with their differences simplified. This error serves as a pivotal factor in selecting the appropriate pixel along the line within the Bresenham line drawing algorithm. It's noteworthy to further expound that in Figure 2B, the coordinates of point M are represented as x, y, where y can be determined from x by the line equation $m(x_{k+1}) + c$. As illustrated in the figure, the coordinates of points M1 and M2 are $[x_{k+1}, y_k]$ and $[x_{k+1}, y_{k+1}]$, respectively. However, due to the integer nature of coordinate representations and pixel indexes, the x-coordinate values vary by one unit between consecutive pixels, as do their respective indexes. Hence, $x_{k+1} = x_{k+1} + 1$.

The parameter Pk captures the discrepancy between the theoretical position of the line and its actual representation on the screen. It is essential for determining the next pixel to render along the line, guiding the algorithm's decision-making process. This Pk represents the initial decision parameter used in the Bresenham line

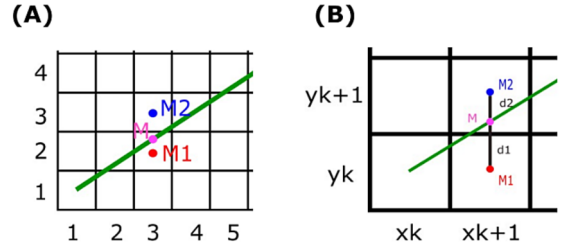

FIGURE 2
How to better understand Bresenham's algorithm. **(A)** A display with a line and three points, M, M1 and M2. **(B)** A display with a line, three points, M, M1, M2 and two distances d1, d2.

1 https://github.com/XQuartz/xorg-server/blob/0ea9b595891f2f31915538192961f3404d9ca699/fb/fbseg.c

drawing algorithm and is crucial for determining the next pixel along the line to render.

$$d1 = y - y_k$$

$$d2 = y_{k+1} - y$$

$$d1 = \left[m\left(x_k + 1\right) + c \right] - yk$$

$$d2 = y_{k+1} - \left[m\left(x_k + 1\right) + c \right]$$

The difference d1 - d2 results in:

$$d1 - d2 = \left[\left(m\left(x_k + 1\right) + c - y_k \right) - \left(y_k + 1 - m\left(x_k + 1\right) + c \right) \right]$$

and, with a simple algebraic manipulation:

$$d1 - d2 = 2m\left(x_k + 1\right) - 2y_k + 2c - 1$$

Substituting d1 - d2 into Bresenham's initial error equation and mΔx for Δy:

$$P_k = \Delta x\left(d1 - d2\right) = \Delta x\left[2m\left(x_k + 1\right) - 2y_k + 2c - 1 \right]$$

$$P_k = 2\Delta y\left(x_k + 1\right) - 2\Delta x y_k + 2\Delta xc - \Delta x$$

This Pk represents the initial decision parameter used in the Bresenham line drawing algorithm and is crucial for determining the next pixel along the line to render. However, the algorithm needs to be recursive, so we need to check Pk + 1, starting with its base value from the difference to Pk:

$$P_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2c\Delta x - \Delta x$$

$$P_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + 2\Delta y + 2c\Delta x - \Delta x$$

$$P_{k+1} - P_k = \left[2\Delta y x_{k+1} - 2\Delta x y_{k+1} \right] - \left[2\Delta y x_k - 2\Delta x y_k \right]$$

$$P_{k+1} - P_k = 2\Delta y\left(x_{k+1} - x_k\right) - 2\Delta x\left(y_{k+1} - y_k\right)$$

Considering that $x_{k+1} = x_k + 1$, we need to factor in that $y_k + 1$ is a variable value based on PK:

$$P_{k+1} = P_k + 2\Delta y\left(x_k + 1 - x_k\right) - 2\Delta x\left(y_{k+1} - y_k\right)$$

As $P_k = \Delta x$ (d1–d2), if the distance d2 is bigger than d1 on Figure 2B (which implies Pk < 0), the best value to represent the new y coordinate would be $y_{k+1} = y_k$, and if d1 > d2, the best value would be $y_{k+1} = y_k + 1$.

So, if Pk < 0:

$$P_{k+1} = P_k + 2\Delta y$$

$$y_{k+1} = y_k$$

else:

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$

$$y_{k+1} = y_k + 1$$

Now we should get the starting value of Pk.

$$P_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta xc - \Delta x$$

For getting the starting value we must first derive the value of c from the line formula:

$$y_1 = mx_1 + c$$

$$c = y_1 - mx_1$$

$$c = y_1 - x_1\left(\frac{\Delta y}{\Delta x}\right)$$

Then substitute it on P1:

$$P_1 = 2\Delta y x_1 - 2\Delta x y_1 + 2\Delta y + 2\Delta x\left[y_1 - x_1\left(\frac{\Delta y}{\Delta x}\right) \right] - \Delta x$$

With a few simplifications we get the initial value of P1:

$$P_1 = 2\Delta y - \Delta x$$

The sequence of deductions is accurate, with the that it's important to note that, in cases where the algorithm deals with lines of small slopes, m < 1. Additionally, since Δy over Δx appears in the denominator in certain equations, it could never be zero. However, this is circumvented by the algorithm's treatment of vertical lines, which allows for the manipulation of coordinates x and y as if they were horizontal lines.

And then there is Bresenham's algorithm. Educationally, the Bresenham algorithm is not just about drawing lines. It's a lesson in optimization, efficiency, and the application of discrete mathematics in solving real-world problems. It demonstrates how a series of intelligent decisions, based on simple arithmetic, can create a visually perfect representation of a line on a pixelated display. For better understanding readers can access videos available online which visually explain the Bradenham's algorithm and how to implement it, such as the one available at this link: https://www.youtube.com/watch?v=CceepU1vIKo.

This approach of breaking down complex problems into manageable steps is a cornerstone of computational thinking and algorithm design, making the Bresenham algorithm a valuable pedagogical tool in computer science education.

7 Code demonstration

To better illustrate how to apply the concepts of vector graphics, coordinates, and transformations to create animations, a simple application employing HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript was described. The concept of a basic HTML structure for the web page is outlined below. HTML is tasked with defining the layout and elements of the project. The HTML includes a canvas element with the id "cartesian-plane," designated for rendering the animation. Additionally, it features an input labeled "BPM:" and a button named "Update BPM," which are intended for setting the animation's speed through user input.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <metaname="viewport"content="width=device-width,initial-
scale=1.0" />
  <title>Smooth Heart Expansion and Contraction</title>
  <!-- CSS Styles -->
  <style>
   body {
    background-color: #121212; /* Dark background */
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
   }
   #labelBpmInput {
    color: white;
   }
   canvas {
    border: 2px solid #fff; /* White border */
   }
  </style>
</head>
<body>
  <canvas id="cartesian-plane" width="400" height="400">
</canvas>
  <label id="labelBpmInput" for="bpmInput">BPM:</label>
  <input type="number" id="bpmInput" min="1" max="240"
step="1" value="60" />
  <button onclick="updateInterval()">Update BPM</
button>
  <script>
   // JavaScript code will be explained below
  </script>
</body>
</html>
```

The CSS styles, also implemented in the example, serve as a language for styling and laying out web pages. CSS dictates the display of HTML elements on screen, with the ability to control a vast array of web page appearance aspects, including font, color, size, spacing, and the layout of elements. Lastly, the JavaScript code is charged with creating and managing the heart's expansion and contraction animation. JavaScript, a programming language designed to add interactivity to the webpage, can facilitate the creation of animations, games, and other interactive elements. This code is articulated through a series of statements, which are directives for the computer's operation. These statements are organized into blocks of code using curly braces ({}).

The function below establishes the Cartesian plane on the canvas, providing a grid and axes for reference. This code segment begins by setting the color palette for the plane and determining the line width. Subsequently, vertical and horizontal lines are drawn to form a grid pattern. Ultimately, the x and y axes are delineated in white.

```
// JavaScript
// Constants for animation scaling
const SCALE_MIN = 1.0;
const SCALE_MAX = 1.2;
const SCALE_STEP = 0.01;
// Variables to control animation state
let scaleFactor = SCALE_MIN;
let expanding = true;
let bpm = 60;
let lastTimestamp = 0;
let interval = 60000 / bpm; // milliseconds per beat
// Function to create the Cartesian plane
function createCartesianPlane() {
  const canvas = document.getElementById("cartesian-plane");
  const ctx = canvas.getContext("2d");
  // Configure color palette for the Cartesian plane
  ctx.strokeStyle = "#ccc"; // Gray
  ctx.lineWidth = 1;
  // Draw vertical lines
  for (let x = 0; x < canvas.width; x += 20) {
   ctx.beginPath();
   ctx.moveTo(x, 0);
   ctx.lineTo(x, canvas.height);
   ctx.stroke();
  }
  // Draw horizontal lines
  for (let y = 0; y < canvas.height; y += 20) {
   ctx.beginPath();
   ctx.moveTo(0, y);
   ctx.lineTo(canvas.width, y);
   ctx.stroke();
  }
  // Draw x and y axes
  ctx.strokeStyle = "#fff"; // White
  ctx.beginPath();
  ctx.moveTo(canvas.width / 2, 0);
  ctx.lineTo(canvas.width / 2, canvas.height);
  ctx.stroke();
```

```
  ctx.beginPath();
  ctx.moveTo(0, canvas.height / 2);
  ctx.lineTo(canvas.width, canvas.height / 2);
  ctx.stroke();
 }
```

The code presented below adjusts the color palette for the lines and fixes the line width. This function processes the coordinates and calculates the pixel positions for line drawing. Following this, it completes the path and executes the line rendering. // Function to draw lines between coordinates

```
function drawLines(coordinates) {
 const canvas = document.getElementById("cartesian-plane");
 const ctx = canvas.getContext("2d");
 // Configure color palette for the lines
 ctx.strokeStyle = "#ff0000"; // Red
 ctx.lineWidth = 3;
 ctx.beginPath();
 // Iterate through the coordinates matrix and draw lines
 for (const [x, y] of coordinates) {
  const centerX = canvas.width / 2;
  const centerY = canvas.height / 2;
  const pointX = centerX + x * 20;
  const pointY = centerY + y * -20; // Negative y-coordinate
  ctx.lineTo(pointX, pointY);
 }
 // Complete the path and draw the lines
 ctx.stroke();
}
```

The constant heartShape represents the array of the coordinates of points that form a heart shape. These points will be scaled to create the animation.

```
// Heart shape represented as a set of coordinates
const heartShape = [
 [0, 1], [1, 2], [2, 2], [3, 1], [3, 0],
 [0, -3], [-3, 0], [-3, 1], [-2, 2], [-1, 2], [0, 1],
];
```

The function below manages the heart expansion and contraction animation. It initiates by clearing the canvas and re-establishing the Cartesian plane. The animation fluctuates between the heart shape's expansion and contraction, influenced by the expanding variable. The scale factor, scaleFactor, is modified accordingly. The heart shape's coordinates are scaled utilizing the scaleFactor. The drawLines function is then invoked to depict the heart shape. The animation ceases when the heart shape reverts to its original size.

```
function animateHeart(timestamp) {
 if (!lastTimestamp || timestamp - lastTimestamp >= interval) {
    // Update the scale factor
    scaleFactor += expanding? SCALE_STEP: -SCALE_STEP;
    if (scaleFactor >= SCALE_MAX || scaleFactor <= SCALE_MIN) {
    expanding = !expanding;
    }
    // Scale the heart shape
    const scaledCoordinates = heartShape.map(([x, y]) => [x * scaleFactor, y * scaleFactor]);
    // Redraw canvas
    createCartesianPlane();
    drawLines(scaledCoordinates);
    lastTimestamp = timestamp;
  }
  requestAnimationFrame(animateHeart);
 }
 // Function to update the BPM and corresponding interval
 function updateInterval() {
  const input = document.getElementById("bpmInput").value;
  bpm = parseInt(input);
  interval = 60000 / bpm;
  lastTimestamp = 0;
 }
```

An illustration of how the HTML animation appears in the first frame can be observed, from which point the heart will enlarge and shrink by 20% based on the entered BPM speed. The complete heart animation is hosted at https://www.cienciaimago.com/heart_imago.html. This article dissected the provided, elucidating the role and functionality of each segment. It also acquainted the reader with the foundational concepts of animations utilizing canvas and JavaScript. Notably, the line drawing algorithm employed bears resemblance to Bresenham's, in that it is supplied with two coordinates and rasterizes a line between them.

8 Discussion

The basic concepts of computer graphics help to understand how animations are created, from writing the initial code to the final product that is shown on the screen. It's important that students and professors understand more deeply what they are seeing, and the process behind that, so they can create their own animations and illustrations to propagate knowledge. A few studies conducted with students already showed that animations can help the understanding of subjects like biology. A quiz with questions on apoptosis was conducted after students saw either the lecture, or the lecture with also an animation about the topic. The results show that the average of correct questions was higher on students that saw both contents. The questions that involved definitions of the subject were not enhanced, but the ones that were about the order or location of the events of apoptosis were (Stith, 2004). Another study was conducted in a biology classroom with the subject of mitosis and meiosis, and the classroom was divided into different groups. As a result, they suggest that the animation significantly improved student retention of the content in students that had the animation with the lecture and as an individual study aid, suggesting that this is the best approach (Veselinovska and Stavreva, 2020). Coluci et al. also states that animations have been used to facilitate learning both in the classroom and in individual study (Coluci, 2022). When it comes to the complexity of the animation, a study on molecular biology showed that even being exposed to a greater visual complexity, students were able to focus on the more thematically

relevant aspects of the animation. Furthermore, it was established that this kind of animation contributed the greatest level of insight into the events depicted. This observation implies that students are more capable of processing visual complexity than the literature on cognitive load would suggest (Jenkinson and McGill, 2013).

Since a few studies showed that animation helps students improve their knowledge in biological processes, the recommendation is that teachers adopt the use of this technology along the traditional ways of teaching to improve students' learning (Awofodu et al., 2022).

However, these studies often lack a critical examination of the underlying mechanisms that make animations effective, such as the mathematical and computational principles involved in their creation. Moreover, they frequently overlook variables like students' prior knowledge of technology, the complexity of the animations, and the potential cognitive load imposed by dynamic visuals.

The current study addresses these gaps by exploring not only the educational benefits of animations in biology but also the mathematical and programming concepts that underpin their development. By elucidating the processes of matrix transformations, rendering algorithms, and graphical programming, this research provides educators and students with a deeper understanding of how animations are constructed and how they can be effectively utilized in teaching complex biological processes. This integrated approach aims to bridge the gap between theoretical knowledge and practical application, offering a more comprehensive perspective on the use of animations in science education and highlighting the role of computer graphics as a vital tool in developing effective educational resources.

9 Conclusion

The use of animation in life science education represents a significant leap forward in how we teach and understand complex biological concepts. Animation, by its very nature, simplifies and visualizes processes that are difficult to grasp through traditional teaching methods. For students, this means that the often invisible or abstract phenomena of biology, such as cellular mechanisms, genetic processes, and ecological interactions, can be made visible and tangible. For educators, animations offer a dynamic way to present biological content. They can break down the steps of a process, like photosynthesis or mitosis, into digestible, sequential animations. This not only aids in retention but also helps students make connections between concepts. Visual learners, in particular, can benefit significantly from animations, but the interactive nature of animated content also engages kinesthetic and auditory learners by providing a multi-sensory learning experience. This inclusivity enhances the overall educational environment, making biology more accessible to a broader range of students.

By revealing the principles behind animation tools—such as vector operations, matrix transformations, coordinate systems, and rendering algorithms—we aim to empower educators and students to not only use animations more effectively but also to critically engage with how they are built. This approach encourages a deeper comprehension of visualized biological processes and opens the door for more meaningful customization and innovation in educational content. Practically, this understanding can enhance the creation of accurate and didactic animations in classrooms, particularly in contexts where access to professional tools is limited.

Future studies could explore the impact of teaching the mathematical and programming principles underlying animations on biology students' comprehension of complex processes. An interesting approach would be exploring how these foundational concepts can be integrated into virtual and augmented reality environments to create immersive and interactive biology learning experiences. Additionally, empirical studies comparing learning outcomes with and without such math-based animation frameworks could help validate their impact and further refine best practices for implementation in science education.

To maximize the benefits of animation in biology education, it is essential to provide clear recommendations. Educators should incorporate animations into curricula to illustrate complex processes, while professional development programs can equip teachers with the skills needed to create and utilize these tools effectively. Content developers must prioritize scientific accuracy, ideally collaborating with subject matter experts. The development and implementation of high-quality animations often require substantial technical expertise and financial resources, which may not be readily available in all educational institutions, particularly those with limited funding. Due to that, policymakers should support initiatives that increase access to animation resources and training, especially in under-resourced settings. Additionally, future studies should consider factors such as cultural relevance and technological accessibility to inform best practices.

In summary, integrating animation into biology education has the potential to transform traditional teaching methods, especially for complex topics. Emphasizing both the use of animation software and the conceptual foundations of computer graphics and mathematics will empower users to fully realize the potential of these tools and shape the future of science education.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

RO: Data curation, Investigation, Software, Validation, Writing – original draft, Writing – review & editing. EA: Data curation, Investigation, Writing – original draft, Writing – review & editing. RS: Writing – review & editing. VC-D-A: Writing – review & editing. JC: Writing – review & editing. LA: Conceptualization, Writing – original draft, Writing – review & editing.

Funding

The author(s) declare that no financial support was received for the research and/or publication of this article.

Acknowledgments

The authors would like to thank the Oswaldo Cruz Institute, CNPQ and FAPERJ for their support.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fcomp.2025.1520930/full#supplementary-material

SUPPLEMENTARY FIGURE 1
Representations of Vectors and Matrices. **(A)** Representations of an unidimensional and a tridimensional vector. **(B)** Representations of a matrix.

SUPPLEMENTARY FIGURE 2
Representation of the diagonal movement of the triangle through the axis by addition of matrices m, n and v which have 3 one dimensional vectors which represent coordinates, resulting in a new matrix m'.

SUPPLEMENTARY FIGURE 3
Representation of the scaling of the triangle by multiplication of the matrix m which has 3 one dimensional vectors which represent coordinates, resulting in a new matrix m'.

SUPPLEMENTARY FIGURE 4
Representation of the difference between the axes of a cartesian grid and a screen.

SUPPLEMENTARY FIGURE 5
Vectors and the color that they represent according to the RGB system. Each of these vectors have 3 indexes, which combined represents a unique color.

SUPPLEMENTARY FIGURE 6
Raster image and how it works. **(A)** A drawing representation of how a raster image works. Each pixel has a color value and so they have a fixed resolution, however there is a problem with said approach, the figure can't be as easily transformed or resized without the loss of quality or characteristics. **(B)** A drawing representation of a raster image in its normal size and after being ampliated.

SUPPLEMENTARY FIGURE 7
SVG image and how it works **(A)** A SVG file of a line and the code that creates it. This shows the content of a basic SVG file and what it renders. **(B)** A SVG drawing of a cell, showing that even if it is ampliated, image quality is not lost.

References

Awofodu, A. D., Ogbonnaya, U., Ogundele, O. E., Zangonde, G. S., and Odusanwo, E. O. (2022). The use of video and cartoon concepts in the teaching and learning of secondary school biology. *Afr. J. Sci. Technol. Math.* 8, 196–201.

Bosch, R., and Heman, A. (2005). Pointillism via linear programming. *UMAP J.* 26, 405–411. doi: 10.2307/27646402

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Syst. J.* 4, 25–30. doi: 10.1147/sj.41.0025

Coluci, V. R. (2022). Animações de Conceitos da Teoria de Erros Usando Manim/Python. *Rev. Bras. Ensino Fis.* 44:e20210239. doi: 10.1590/1806-9126-RBEF-2021-0239

Eisenberg, J. D. (2002). SVG Essentials. O'Reilly.

Ibraheem, N. A., Hasan, M. M., Khan, R. Z., and Mishra, P. K. (2012). Understanding color models: a review. *ARPN J. Sci. Technol.* 2, 265–275.

Jenkinson, J., and McGill, G. (2013). Using 3D animation in biology education: examining the effects of visual complexity in the representation of dynamic molecular events. *J. Biocommun.* 39, 42–49.

Kalas, P., and Redfield, R. J. (2022). Using animations to teach biological processes and principles. *PLoS Biol.* 20:e3001875. doi: 10.1371/journal.pbio.3001875

Kist, G. (2020). Abordagem Da Utilização Da Álgebra Linear Em Transformações Geométricas Implementadas Na Visão Computacional. Rio Grande do Sul: Universidade do Vale do Taquari - UNIVATES.

Nayyer, R., and Sharma, B. (2015). Use and analysis of color models in image processing. *Int. J. Adv. Sci. Res.* 1, 329–330. doi: 10.7439/ijasr.v1i8.2460

O'Day, D. H. (2007). The value of animations in biology teaching: a study of long-term memory retention. *CBE Life Sci. Educ.* 6, 217–223. doi: 10.1187/cbe.07-01-0002

Peart, D. J., Keane, K. M., Allen, G., Bruce-Martin, C., and Rumbold, P. L. S. (2022). Using animations to support student learning in undergraduate physiology. *J. Biol. Educ.* 56, 432–442. doi: 10.1080/00219266.2020.1821082

Praveen, C. K., and Srinivasan, K. (2022). Psychological impact and influence of animation on viewer's visual attention and cognition: a systematic literature review, open challenges, and future research directions. *Comput. Math. Methods Med.* 2022:8802542. doi: 10.1155/2022/8802542

Shishmanova, S., and Rinaldi, A. (2018). RGB color wheel intended to create color harmony compositions in modern art and design. *EPH - Int. J. Sci. Eng.* 4, 45–57. doi: 10.53555/eijse.v4i4.163

Stith, B. J. (2004). Use of animation in teaching cell biology. *Cell Biol. Educ.* 3, 181–188. doi: 10.1187/cbe.03-10-0018

Veselinovska, S. S., and Stavreva, A. (2020). The impact of the usage of web animation in teaching molecular and cellular biology. *J. Educ. Sci. Theory Pract.* 11, 116–127. doi: 10.46763/jespt

Wildhaber, R. A., Verrey, F., and Wenger, R. H. (2011). A graphical simulation software for instruction in cardiovascular mechanics physiology. *Biomed. Eng. Online* 10, 1–6. doi: 10.1186/1475-925X-10-8

Zelazko, A. (2023). RGB colour model. Encyclopedia Britannica. Available online at: https://www.britannica.com/science/RGB-colour-model (Accessed August 24, 2023).