ПРИКЛАДНА МАТЕМАТИКА

УДК 519.6: 004.94
DOI https://doi.org/10.32782/2521-6643-2024-2-68.1

Pasichnyk A. M., Doctor of Physical and Mathematical Sciences, Professor, Professor at the Department of Mathematical Modeling and System Analysis
Dniprovsky State Technical University
ORCID: 0000-0002-8561-1374

Nadryhailo T. Zh., Candidate of Technical Sciences, Docent, Associate Professor at the Department of Mathematical Modeling and System Analysis
Dniprovsky State Technical University
ORCID: 0000-0003-1239-5946

Andrieiev O. V., Graduate student
at the Department of Mathematical Modeling and System Analysis
Dniprovsky State Technical University

A COMPREHENSIVE IMPLEMENTATION ALGORITHM 2D TRANSFORMATION IN GRAPHIC WEB-EDITORS BASED ON THE METHODS OF AFFINE TRANSFORMATIONS

This article presents a complex algorithm for the optimized implementation of 2D transformations in graphic web editors based on the methods of affine transformations.

The object of this study is the algorithms and methods of affine transformations for the implementation of 2D transformations in graphic web editors. The research is aimed at applying the principles of these transformations and their impact on graphic objects, as well as at finding effective solutions for their implementation.

An analysis of the main existing algorithms and methods for implementing 2D transformations, such as the Bresenham and Wu algorithms for visualizing straight and curved lines, algorithms for filling areas using affine transformation methods, was carried out. Therefore, the issue of improving and simplifying algorithms and methods of affine transformations for implementing 2D transformations in graphic web editors is relevant and important for modern web design and development.

The work considers the main algorithms of affine transformations that ensure the implementation of scaling, rotation, shift, skew functions and their interaction with graphic objects. The proposed algorithm includes the main advantages of implementing the above functions and allows you to create more efficient and high-quality graphic applications, providing at the same time greater functionality and productivity. Applying such an approach creates new opportunities for the development and use of new types of web editors in the field of web design.

Based on the proposed algorithm, a web application has been developed that allows you to create various transformations and animations of figures on the plane. The results of the conducted testing confirm the effectiveness and possibility of applying the proposed approach in real projects for the development of graphic web editors for working with 2D transformations.

Key words: *affine transformations, graphic editor, 2D transformations, web- editor, transformations in CSS, matrix decomposition.*

Пасічник А. М.,Надригайло Т. Ж., Андрєєв О. В. Комплексний алгоритм реалізації 2d-трансформацій в графічних веб-редакторах на основі методів афінних перетворень

В даній статті представлено комплексний алгоритм оптимізованої реалізації 2D-трансформацій в графічних веб-редакторах на основі методів афінних перетворень.

Об'єктом даного дослідження є алгоритми та методи афінних перетворень для реалізації 2D-трансформацій у графічних веб-редакторах. Дослідження спрямоване на застосування принципів цих трансформацій та їх впливу на графічні об'єкти, а також на пошук ефективних рішень для їх реалізації.

Проведено аналіз основних існуючих алгоритмів та методів реалізації 2D-трансформацій, таких як алгоритми Брезенхема та Ву для візуалізації прямих та кривих ліній, алгоритми заповнення областей, що використовують методів афінних перетворень. Тому питання удосконалення та спрощення алгоритмів та методів афінних перетворень для реалізації 2D-трансформацій в графічних веб-редакторах є актуальним та важливим для сучасного веб-дизайну та розробки.

В роботі розглянуті основні алгоритми афінних перетворень, що забезпечують реалізацію функцій масштабування, повороту, зсуву, перекосу та їх взаємодію з графічними об'єктами. Запропонований алгоритм включає основні переваги реалізації наведених функцій та дозволяє створювати більш ефективні та високоякісні графічні додатки, забезпечуючи при цьому більшу функціональність та продуктивність. Застосування такого підходу створює нові можливості для розробки та використання нових типів веб-редакторів в галузі веб-дизайну.

На основі запропонованого алгоритму розроблено веб-застосунок за допомогою якого можна створювати різноманітні перетворення та анімації фігур на площині. Результати проведеного тестування підтверджують ефективність та можливість застосування запропонованого підходу у реальних проектах з розробки графічних веб-редакторів для роботи з 2D-трансформаціями.

Ключові слова: *афінні перетворення, графічний редактор, 2D трансформації, веб–редактор, перетворення у CSS, декомпозиція матриці.*

Formulation of the problem. Affine transformations are a key element of computer graphics, especially in the field of creating and manipulating 2D images and animations. Web-based graphic editors are one of the key tools for creating and sharing digital content on the Internet, such as images, animations, videos, and interactive elements. The effectiveness of the development of relevant editors is determined by the ability to create complex high-quality graphic objects and effects with minimal load on system resources. In this regard, to improve the performance and quality of graphics interpretation in web editors, further improvement and optimization of the use of algorithms and methods of affine transformations for the implementation of 2D transformations are of current and practical importance.

Analysis of recent research and publications. Athenian transformations are mathematical operations that allow you to implement image processing with changes in the shape, size, position, and skew of graphic objects in two-dimensional space. They are used in graphic web editors to implement 2D transformations and graphic effects. An affine transformation changes the coordinates of each point of an object by multiplying its coordinates by a transformation matrix and adding a shift vector [1]. The main types of affine transformations are used to scale, rotate, shift, skew, and translate images. At the same time, affine transformations preserve straight lines and parallelism, the distance between points and the middle of a segment, proportions between lengths, and angles between vectors. These properties of affine transformations ensure that objects are resized and positioned correctly in web-based graphic editors.

Most editors use the CSS coordinate space with a two-axis coordinate system: the X-axis increases horizontally to the right; the Y-axis increases vertically downward [2]. To accumulate transformation elements, a local coordinate system is used, where an element actually accumulates all the transformation properties of its predecessors, as well as any local transformation applied to it. The accumulation of these transformations determines the current CTM transformation matrix for the element, which is calculated by multiplying all transformation matrices, starting from the viewport coordinate system and ending with the element transformation matrix [3].

The matrix multiplication algorithm is one of the main methods for implementing affine transformations. This approach is effective and versatile for implementing various types of transformations and is implicitly supported by native web browser tools, which greatly simplifies the development, support, and integration into existing projects with low-level optimization of the algorithms used in this approach [3-5].

To implement linear transformations, such as shift, rotation, and scaling, the Bresenham algorithm, based on the idea of coordinate increment and pixel change based on comparison with certain thresholds, has been widely used. This algorithm is fast and accurate for implementing simple linear transformations [6]. Later, it was generalized to construct second-order curves [7].

There is also a well-known constraint rectangle algorithm used to optimize the image of objects during transformations. The basic idea is to use bounding rectangles (or "bounding rectangles") for each object or group of objects [8]. The use of the constraint rectangle algorithm can improve the performance of displaying complex scenes, especially during object transformations and visualization [9].

The decomposition algorithm is based on classical LU and QR decompositions. Using CSS transformations: CSS (Cascading Style Sheets) provides built-in functions for performing various 2D transformations directly on HTML elements and provides scaling, rotation, shift, and skew transformations. CSS transformations are easy to use, allow you to quickly achieve the desired effect, but their capabilities are limited and more complex transformations may require additional logic and the use of JavaScript [10].

There are also well-known graphic tools such as SVG (Scalable Vector Graphics) and Canvas that implement a number of 2D transformation functions that provide more flexible and detailed control over transformations, including matrix and other operations on individual elements. These libraries usually require the use of JavaScript for the programmatic implementation of transformations and element manipulation [11].

In this paper, to develop effective graphical web editors that provide higher accuracy and performance of image transformations, we propose to use a comprehensive algorithm based on affine transformations that realizes the main advantages of the algorithms mentioned in the above analysis.

The purpose of the article is to develop a comprehensive algorithm for implementing 2D-transformations in graphic web-editors using affine transformations, which includes the main advantages of the above algorithms.

Presenting main material. 1 The web editor developed in accordance with the proposed concept implements the following functionality: adding 4 basic shapes to the canvas: rectangle, triangle, star and ellipse; deleting shapes; each shape has the following attributes: position (offset): along the *x* and *y* axes; size: width and height; rotation angle; slope angle; transparency. All attributes are available for change in the properties panel; it is possible to drag, resize, and rotate the figure manually using the appropriate graphical interface elements (manipulator), without entering a specific numerical value in the attribute fields; animation of figures. Animation, in addition to animated values, also contains a delay, duration, and a time-dependent function for realizing various effects and smoothness of animations. The following attributes are available for animation: position (offset): along the *x* and *y* axes; scaling: in the horizontal and vertical directions; rotation angle; tilt angle; transparency. Shape animation does not affect the behavior of the manipulator; it can be used to change shape attributes. You can play, reset, and view the total animation time. Shapes are animated separately from each other, but playback and animation time are common to all from the user's point of view.

TypeScript (*JavaScript*) was chosen to create the graphical web editor because it is a standard for developing client applications and has clear advantages over other technologies.

To optimize the development and deployment of the project, it is proposed to use modern *JavaScript* frameworks such as *Vue*, *React*, and *Angular* [10]. The chosen technologies provide solutions to a number of problems that arise in application development: data reactivity; global application state; architectural problems such as dividing the application into components/modules and code cleanliness; optimization of visualization and re-visualization when making many changes to the DOM; declarative syntax, etc.

Since the program is a graphical editor, it is also important to provide an attractive and user-friendly graphical interface. For the convenience of the graphical interface we chose the *PrimeVue* library, which has a fairly large set of ready-made user interface components, as well as because of its simplicity and rational documentation.

To simplify working with matrices and calculating basic matrix operations, we chose the *mathjs* library. To simplify the creation of animations, we chose the *animejs* library, which has high performance and is a fairly low-level tool for flexible animation of CSS properties and JS objects.

To implement the image of shapes on the canvas, a stylized *div* block is used. The entire component consists of two key blocks: an SVG component that will represent the shape without transformations, and a *div* that prepares it for applying affine transformations to the component. Before proceeding with transformations, you need to build an SVG path for each shape.

In the case of a rectangle, 4 points are defined based on the position and dimensions of the shape: top-left – (0, 0); top-right – (width, 0); bottom-right – (width, height); bottom-left – (0, height). The result is the image shown in Figure 1.

Similarly, for a triangle: top-center – (width / 2, 0); bottom-right – (width, height); bottom-left – (0, height). The result is the image shown in Figure 2.

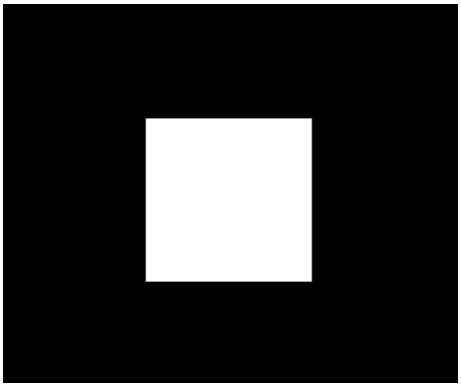

Figure 1. Graphical display of the calculated rectangle

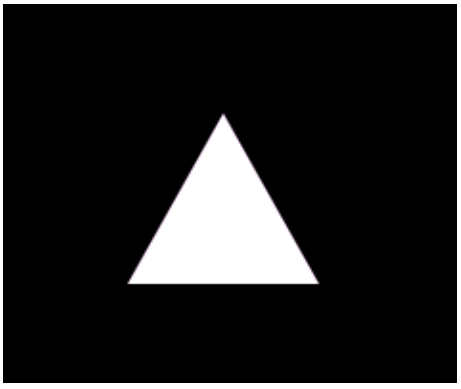

Figure 2. Graphical display of the calculated triangle

In the case of an ellipse, the algorithm will be slightly different, since 2 elliptic curves must be used instead of straight lines according to the algorithm: start at point (0, height /2); a large elliptic curve with $r_x = width /2$, $r_y = height /2$ is introduced at point (width, height /2); and a large elliptic curve with $r_x = width /2$, $r_y = height /2$ at point (0, height /2).

The result is the image shown in Figure 3.

For a star, the algorithm will be a little more complicated than for other shapes, since you need to correctly calculate the points for all 10 corners of the star.

First, the length of the inscribed circle of the star and its radius are calculated:

$$circumference = \min\left(width, height\right), \qquad r = \frac{circumference}{2}. \tag{1}$$

Then the scaling factors are calculated, which determine the degree of stretching of the star along the *x* and *y* axes:

$$s_x = \frac{width}{circumference}, \qquad s_y = \frac{height}{circumference}. \tag{2}$$

The angles for each vertex of the star are calculated using the following formulas:

$$angle_i = i * \frac{\pi}{5} - \frac{\pi}{2}, \tag{3}$$

where *i* is the index of the star vertex from 0 to 9.

Next, for each vertex angle, the coordinates of the point (*x*, *y*) on the circle are calculated and the star is scaled using the following formulas:

$$x_i = \left(r + r_i * \cos angle_i\right) * s_x, \tag{4}$$

$$y_i = \left(r + r_i * \sin angle_i\right) * s_y, \tag{5}$$

where $r_i = r$ for even values of *i* (outer vertices) and $r_i = r * 0.382$ for odd values (inner vertices). The value of 0.382 is determined from the condition of proportionality of the indents between the outer and inner vertices of the star. A graphical representation of the star using the algorithm is shown in Figure 4.

All these shapes are SVG paths without transformations.

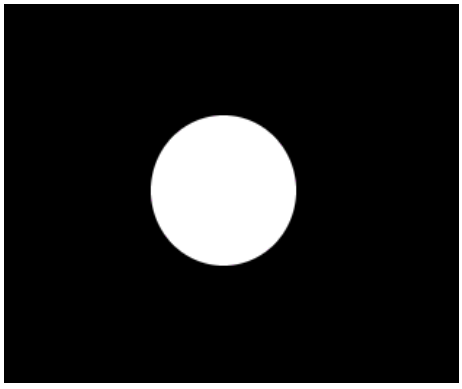

Figure 3. Graphical display of the calculated ellipse

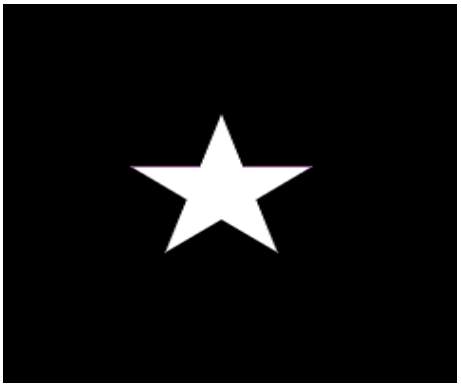

Figure 4. Graphical display of the calculated star

The matrices of individual transformations are defined as follows:

$$translation\left(x, y\right) = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} \tag{6}$$

$$scale\left(x, y\right) = \begin{pmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

$$rotation\left(\theta\right) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8}$$

$$skew\left(x, y\right) = \begin{pmatrix} 1 & \tan x & 0 \\ \tan y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{9}$$

Since the decomposed components of the transformation matrix are used in a number of calculations, the algorithm for the reverse composition of the decomposed elements is written as follows:

$$TM = translation(x, y) * rotation(\theta) * scale(scaleX, scaleY) *$$

$$*skew(skewX, skewY) \tag{10}$$

To transform the shape attributes into 4 basic ones, first, the position of the shape is transformed as if the origin was in the center of the shape:

$$x_c = x + \frac{width}{2} \tag{11}$$

$$y_c = y + \frac{height}{2} \tag{12}$$

With this choice, all subsequent transformations, except for the shift, will be calculated relative to the center of the shape, and the shape transformation matrix is calculated as follows:

$$TM_s = translation(x_c, y_c) * rotation(\theta_s) * scale(scaleX_s, scaleY_s) *$$

$$*skew(skewX_s, skewY_s) \tag{13}$$

Applying a transformation matrix to shapes is implemented using the CSS *transform* attribute with the value *matrix*(*a*, *b*, *c*, *d*, *tx*, *ty*).

Since the transformations are calculated relative to the center of the shape, the image block must be shifted by half the horizontal width and half the vertical height.

The next step is to implement a panel for changing shape attributes. On the panel, we need the following attributes: position, size, rotation, skew, and opacity. Also, for the convenience of canceling the values of all attributes, if necessary, the "Clear" button is added. A general view of the attributes panel is shown in Figure 5.

The manipulator is responsible for dragging, resizing, and rotating shapes on the canvas with the mouse, that is, without using the attributes panel. In this case, it is enough to remember the initial position of the component (i.e., at the time of the start of dragging), and then add the mouse offset to it relative to the point of the start of dragging:

$$x_{new} = x_{initial} + x_{mouse} \tag{14}$$

$$y_{new} = y_{initial} + y_{mouse} \tag{15}$$

All shape transformations are taken into account to ensure that the original position of the element is maintained after resizing. Eight "labels" are introduced to resize by dragging in the appropriate directions. To implement the "labels" themselves, you use a regular *div* with the appropriate positioning on the boundaries of the bounding rectangle of the figure for each "label".

To drag any of the "labels" 2 decomposed transformations are used:

1. The decomposed TM_{shape} matrix without animations, i.e. the shape's own transformations.
2. The decomposed TM_{full} matrix with animations, i.e. the shape transformations overridden by animations at a specific moment in the animation.

The next step is to implement 4 "labels" for the shape rotation, one for each corner of the shape bounding box. To calculate the rotation while dragging the "labels", you need to save the shape transformation matrix, taking into account the animations at the time of the start of the drag, as well as the initial rotation angle (without animations) and the angles of the shape bounding box, taking into account all the animations.

First, the angles of the shape's bounding box are calculated. To do this, the transformation matrix is converted to the original coordinate system:

$$TM_{source} = TM_{full} * translation\left(-\frac{width_{full}}{2}, -\frac{height_{full}}{2}\right) \tag{16}$$

and then it is applied to the initial points of the bounding box:

$$\begin{pmatrix} x_{topLeft} \\ y_{topLeft} \\ 1 \end{pmatrix} = TM_{source} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} x_{topRight} \\ y_{topRight} \\ 1 \end{pmatrix} = TM_{source} \begin{pmatrix} width_{full} \\ 0 \\ 1 \end{pmatrix}, \tag{17}$$

$$\begin{pmatrix} x_{bottomLeft} \\ y_{bottomLeft} \\ 1 \end{pmatrix} = TM_{source} \begin{pmatrix} 0 \\ height_{full} \\ 1 \end{pmatrix}, \tag{18}$$

$$\begin{pmatrix} x_{bottomRight} \\ y_{bottomRight} \\ 1 \end{pmatrix} = TM_{source} \begin{pmatrix} width_{full} \\ height_{full} \\ 1 \end{pmatrix}. \quad (19)$$

Next an algorithm for calculating the new angle of rotation of the shape each time you drag the "label" is implemented:

1. The position of the dragged mark is determined from the previously calculated manipulator angles.
2. Calculates the position of the "label" without taking into account the offset:

$$\begin{pmatrix} x_{origin} \\ y_{origin} \\ 1 \end{pmatrix} = translation\left(-x_{full}, -y_{full}\right) \begin{pmatrix} x_{handle} \\ y_{handle} \\ 1 \end{pmatrix}. \quad (20)$$

3. The current angle is found relative to the dragged point. The function *atan2*(*y*, *x*) returns the angle between the positive *x*-axis and the ray from (0, 0) to (*x*, *y*):

$$angle_{current} = atan2\left(y_{origin}, x_{origin}\right). \quad (21)$$

4. The new angle is found relative to the dragged point, taking into account the mouse offset:

$$angle_{new} = atan2\left(y_{origin} + y_{mouse}, x_{origin} + x_{mouse}\right). \quad (22)$$

After implementing the manipulator logic, we can move on to implementing animations. Each animation has the following attributes:

- Delay before the start and duration of the animation.
- The distance between the start and end coordinates of the figure to animate the movement of the figure along the *x* and *y* axes.
- The *x* and *y* scale factors for animating the shape's resizing.
- Rotation.
- The skew: x та y.
- Opacity.

The panel for changing the animation attributes of the selected component will look like the one shown in Figure 6.

The *animejs* library is used to implement animations. The animation attributes are converted to a format that can be written to the *animejs* library to animate the elements. To do this, the initial and final values of the animation and the animated transformation are calculated.

In total, we have 8 types of animated transformations, including transparency. The following algorithm is used to calculate the start and end values of animations:

- The initial value is always equal to the current value of the corresponding shape attribute, except for scaling, since shapes don't have their "own" scaling attribute. Therefore, the initial value in this case is always 1.
- The final value is always equal to the corresponding animation attribute, except for shift and rotates, because the values of these transformations are represented relative to the shape's transformations. Therefore, the final value in this case is equal to the sum of the values of the corresponding shape and animation attribute.

Next, the list of animations is passed to the *animejs* library with the specified delay, duration, start and end values. For simplicity, the function from the animation time is linear for all animations.

The following algorithm, which uses the *animejs* library to calculate the numerical value of a particular animation at the current time, is as follows:

1. Call the *requestAnimationFrame*(*callback*) function [11] with the scheduled animation step if the time from the start of the animation does not exceed its duration. This function tells the browser that you want to perform an animation and you request that the browser call the specified function to update the animation right before the next redraw. The callback frequency is 60 times per second, which corresponds to the display refresh rate in most web browsers as recommended by the W3C technology standards [8].
2. Calculate the time from the beginning of the animation and its ratio to the total duration, i.e. the coefficient from 0 to 1.
3. Pass the coefficient to the easing function, which is specified by the user and returns a new coefficient in the same format. In our case, the coefficient will not be changed, since we are using a linear function.
4. Multiply the coefficient obtained in the previous step by the difference between the final and initial animation values and adds the result to the initial animation value.
5. Repeat step 1.

To visualize each figure, a separate local animation complex with common control is implemented. The results of transforming images of these types using the proposed approach are shown in Figure 7.

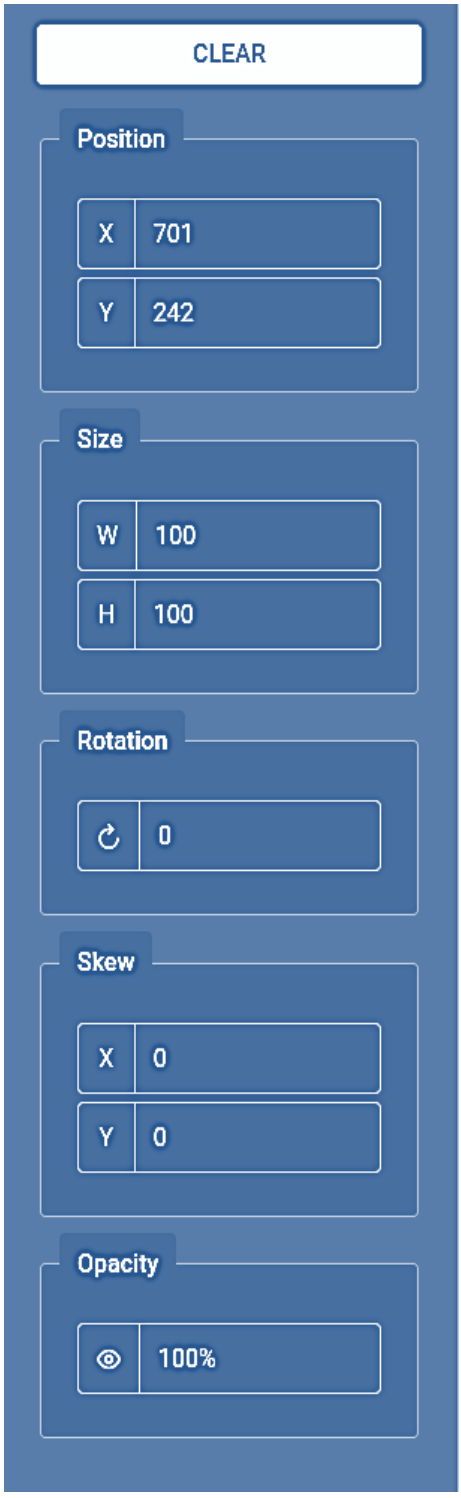

Figure 5. Graphical representation of the attributes panel for shapes

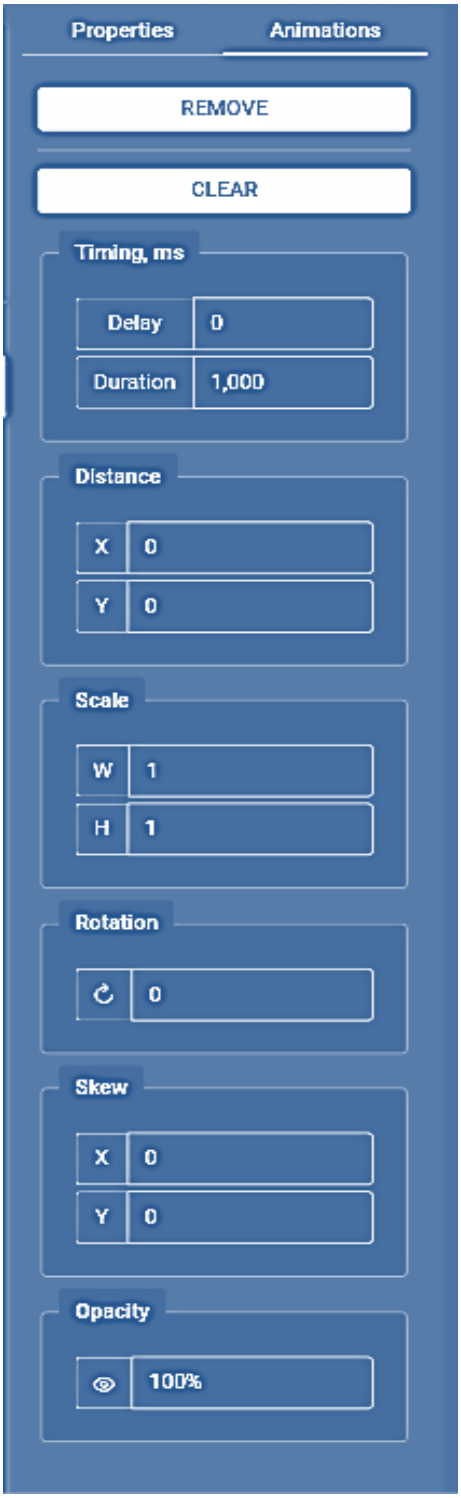

Figure 6. Graphical representation of the animation attributes panel

In addition, we also need to implement partial replacement of component attributes with new transformations at a certain point in the animation playback time to display them on the canvas and use them in the manipulator calculations. The end result is a convenient and attractive graphical web editor with the ability to transform and animate shapes, which has excellent performance.

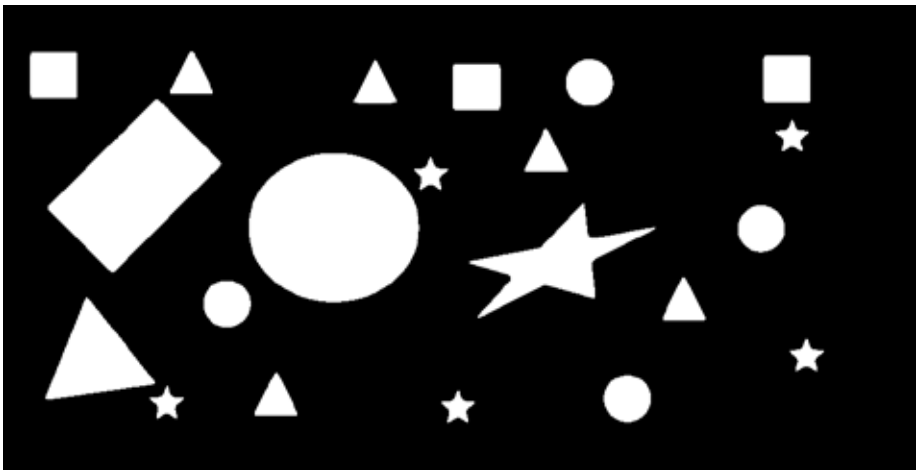

Figure 7. The result of image transformations implemented in software

Conclusions. Based on the analysis of the main existing algorithms and methods for implementing 2D transformations, a comprehensive algorithm for the optimized implementation of 2D transformations in graphic web editors based on affine transformations that provide the implementation of scaling, rotation, shift, and skew functions has been developed. The proposed algorithm incorporates the main advantages of implementing these functions and allows creating more efficient and high-quality graphic applications, while providing greater functionality and performance. The application of this approach creates new opportunities for the development and use of new types of web editors in the field of web design.

Bibliography:

1. Algorithm Archive – Affine Transformations. [Електронний ресурс]. – Режим доступу: https://www.algorithm–archive.org/contents/affine_transformations/affine_transformations.html

2. W3Schools – CSS Transforms[Електронний ресурс]. – Режим доступу: https://www.w3schools.com/css/css3_2dtransforms.asp.

3. MDN Web Docs – transform–origin [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en–US/docs/Web/CSS/transform–origin.

4. Tutorialspoint – Computer Graphics 2D Transformation. [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/computer_graphics/2d_transformation.htm.

5. Frederic Wang. Decomposition of 2D–transform matrices. [Електронний ресурс]. – Режим доступу: https://frederic–wang.fr/decomposition–of–2d–

6. Wikipedia – Bresenham's line algorithm. [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.transform–matrices.html

7. James D. Foley. Computer Graphics: Principles and Practice 2nd Edition / James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes // The System Programming Series, Addison-Wesley Publishing Company, 1996, 1250 p.

8. Addison–Wesley W3C Editor's Draft – Chapter 8: Coordinate Systems, Transformations and Units. [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/SVG/coords.html.

9. Wikipedia – Bounding volume. [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Bounding_volume.

10. VueJS Framework Documentation. [Електронний ресурс]. – Режим доступу: https://vuejs.org/.

11. MDN Web Docs – Window: requestAnimationFrame() method. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en–US/docs/Web/API/window/requestAnimationFrame

References:

1. Algorithm Archive – Affine Transformations. URL: https://www.algorithm–archive.org/contents/affine_transformations/affine_transformations.html.

2. W3Schools – CSS Transforms URL: https://www.w3schools.com/css/css3_2dtransforms.asp.

3. MDN Web Docs – transform–origin. URL: https://developer.mozilla.org/en–US/docs/Web/CSS/transform–origin.

4. Tutorialspoint – Computer Graphics 2D Transformation. URL: https://www.tutorialspoint.com/computer_graphics/2d_transformation.htm

5. Frederic Wang – Decomposition of 2D–transform matrices. URL: https://frederic–wang.fr/decomposition–of–2d–transform–matrices.html.

6. Wikipedia – Bresenham's line algorithm. URL: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.
7. Foley, J.D., van Dam A., Feiner S.K., Hughes J.F. (1996). Computer Graphics: Principles and Practice 2nd Edition / The System Programming Series, Addison-Wesley Publishing Company, 1250 p.
8. Addison–Wesley W3C Editor's Draft – Chapter 8: Coordinate Systems, Transformations and Units. URL: https://www.w3.org/TR/SVG/coords.html.
9. Wikipedia – Bounding volume. URL: https://en.wikipedia.org/wiki/Bounding_volume.
10. VueJS Framework Documentation. URL: https://vuejs.org/.
11. MDN Web Docs – Window: requestAnimationFrame() method. URL: https://developer.mozilla.org/en–US/docs/Web/API/window/requestAnimationFrame