https://doi.org/10.31891/2219-9365-2026-86-28
UDC 004.92:519.6

KRYVDA Pavlo
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
https://orcid.org/0009-0006-8903-059X
pav.kryv@ukr.net
SULEMA Olga
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
https://orcid.org/0000-0001-6450-0993
sulema.olga@lll.kpi.ua

ALGORITHMIC AND SOFTWARE SUPPORT FOR MULTI-CRITERION OPTIMAL TRIANGULATION OF POLYGONS TO INCREASE THE EFFICIENCY OF 3D VISUALIZATION SOFTWARE SYSTEMS

The article considers the problem of increasing the productivity of three-dimensional visualization software systems by optimizing the process of triangulation of polygonal models. Most modern graphics engines use triangles as basic rendering primitives, however, standard triangulation algorithms are focused mainly on the speed of mesh construction, rather than on the subsequent efficiency of visualization. A method of multi-criteria selection of the optimal option for triangulation of a simple polygon is proposed based on the evaluation of all admissible partition structures or their reduced set. The optimization criteria used are minimization of the area of pixel repainting, improvement of the geometric shape of triangles, and reduction of computational costs during rendering. A software model of the algorithm is developed using dynamic programming. Experimental modeling results are presented, which confirm the possibility of reducing redundant graphic operations and increasing the efficiency of displaying complex scenes.
Keywords: algorithmic support, software, triangulation, rendering, optimization, computer graphics, 3D visualization.

КРИВДА Павло, СУЛЕМА Ольга
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМАЛЬНОЇ ТРИАНГУЛЯЦІЇ ПОЛІГОНІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНИХ СИСТЕМ 3D-ВІЗУАЛІЗАЦІЇ

У статті розглянуто проблему підвищення продуктивності програмних систем тривимірної візуалізації шляхом оптимізації процесу тріангуляції полігональних моделей. Більшість сучасних графічних рушіїв використовують трикутники як базові примітиви рендерингу, однак стандартні алгоритми тріангуляції орієнтовані переважно на швидкість побудови сітки, а не на подальшу ефективність візуалізації. Запропоновано метод багатокритеріального вибору оптимального варіанта тріангуляції простого полігона на основі оцінювання всіх допустимих структур розбиття або їхньої скороченої множини. Як критерії оптимізації використано мінімізацію площі повторного зафарбовування пікселів, покращення геометричної форми трикутників і зменшення обчислювальних витрат під час рендерингу. Розроблено програмну модель алгоритму з використанням динамічного програмування. Наведено результати експериментального моделювання, які підтверджують можливість зменшення кількості надлишкових графічних операцій і підвищення ефективності відображення складних сцен.
Ключові слова: алгоритмічне забезпечення, програмне забезпечення, тріангуляція, рендеринг, оптимізація, комп'ютерна графіка, 3D-візуалізація.

Стаття надійшла до редакції / Received 15.04.2026
Прийнята до друку / Accepted 12.05.2026
Опубліковано / Published 31.05.2026

INTRODUCTION

In the modern world, there is an increase in visual communications, textual perception of meaning is giving way to its visually active understanding: computer graphics, hypertexts of electronic manuals, visual models of electromechanical complexes, as well as a wide range of scientific research, the results of which cannot be expressed in verbal form, which requires the development of new methodologies for modeling visual perception. The most informative are three-dimensional (3D) images, since they allow you to evaluate the constructive and aesthetic features of objects. Systems for synthesizing realistic images must ensure the transmission of all properties of the modeled object: volume, location, transmission of halftones, shadows, lighting, surface texture. The higher the degree of realism of the image, the more calculations are required for its formation. Visualization is the final stage of work on the scene being modeled. At this stage, the computer converts the mathematical model of the scene into a form accessible to visual perception. This process is called rendering.

The main difficulties that arise during the execution of the 3D visualization process are a large number of calculations of the coordinates of the position of 3D objects, calculations of the set of pixels on the screen that will be used during output, calculations of the triangulation process of a 3D object, calculations of the process of removing invisible surfaces, and a relatively low speed of image output. Complex images are formed from fragments of objects,

for which they are divided into components. The most common method used is to divide images into triangles. This is due to the following reasons:

- a triangle is the simplest polygon, the vertices of which uniquely define a face;
- any area can be guaranteed to be divided into triangles;
- the computational complexity of the triangulation algorithms is much lower than when using other polygons;
- the implementation of rendering procedures is simplest for an area bounded by a triangle;
- for a triangle, it is easy to determine its three closest neighbors that share common faces with it.

The process of dividing a polygonal region with a complex configuration into a set of triangles is called triangulation. In computer graphics, the most common triangulation algorithms are triangulation of polygonal regions and triangulation of a set of points. Among the methods of triangulation of a set of points defining a surface, the Delaunay method is widely used, which involves dividing a set of points into a set of triangles such that no point from the set lies inside the circle circumscribed around any triangle, except for the vertices of the triangle itself. When analyzing or synthesizing complex surfaces, they are approximated by a mesh of triangles, and then operated on with the simplest polygonal regions, that is, with each of the triangles.

The particular interest in triangulation algorithms is determined by the fact that they are used in many computer graphics procedures, such as surface shaping, coloring, removal of invisible parts, and clipping.

RELATED WORK

The problem of triangulation of polygonal regions remains one of the key ones in computer graphics, geometric modeling, computer-aided design, and real-time systems. Classical methods, in particular Delaunay triangulation [1], ear clipping, sweep-line, and divide-and-conquer algorithms, continue to be used as basic ones, but modern research is aimed at improving the performance, adaptability, and quality of the constructed meshes. In particular, recently, research has shifted from classical geometric algorithms to intelligent, GPU-accelerated, and application-oriented methods. This is due to the increasing demands on rendering speed, numerical modeling, digital twins, and interactive 3D visualization. Thus, in [2], the MeshingNet method was proposed, in which a neural network predicts the local density of mesh elements for the subsequent generation of a high-quality unstructured mesh. The authors showed the possibility of automatically generating high-quality meshes for arbitrary polygonal geometries and various PDE problems, which indicates the prospects for integrating artificial intelligence into the triangulation and mesh generation problem.

The authors of the article [3] formulate the mesh generation problem as a globization optimization problem based on physically informed neural networks. The results obtained confirmed the possibility of fast generation of high-quality structured meshes for complex areas.

The article [4] presents the MeshGPT model, which uses transformers for autoregressive generation of triangular mesh models. The authors report an improvement in the shape coverage index by 9% and a significant increase in the FID quality of the models compared to previous approaches.

The work [5] is devoted to the current state of Delaunay triangulation algorithms for CPU, GPU and parallel architectures. The study shows that the Delaunay method remains the basic standard for many applications, but needs to be adapted to modern high-performance computing platforms.

In [6], it was proposed to use curvilinear boundaries of triangles instead of linear ones. This allowed to reproduce image contours more accurately and reduce the number of mesh elements without losing the quality of visualization.

The authors of [7] proposed to use reinforcement learning to automatically improve the quality of the triangular Delaunay mesh. This confirms the trend of transition from static algorithms to self-learning mesh generation systems.

In [8], PI-MeshONet, a self-learning universal approach to mesh generation for numerical modeling, is proposed. The authors position it as a generalizable/self-supervised method suitable for various types of problems.

In [9], an algorithm for direct rendering of internal triangulations without constructing intermediate subdivisions is proposed. This is especially important for real-time systems.

In the analytical article [10], 113 scientific papers on intelligent mesh generation are systematized. The authors note that triangular meshes remain the dominant format for representing surfaces due to their simplicity, flexibility, and broad software support. At the same time, the lack of universal solutions for the optimal partitioning of arbitrary polygons is emphasized.

Thus, an analysis of the literature shows that research on triangulation optimization is carried out in the following areas: the use of artificial intelligence for constructing meshes; the use of reinforcement learning for optimizing Delaunay algorithms; GPU and parallel implementations; differentiable triangulation; direct rendering of triangular structures; multi-criteria optimization of triangle quality.

At the same time, most modern works are focused either on the quality of the mesh or on the speed of construction, but do not sufficiently take into account the efficiency of subsequent polygon rendering. That is why the

task of finding the optimal triangulation option based on the criteria of minimizing the repainting area, geometric quality of triangles, and visualization speed remains relevant.

PROBLEM STATEMENT

Modern three-dimensional graphics software systems are widely used in computer games, CAD/CAM systems, virtual and augmented reality, digital twins, simulators, and scientific modeling. Regardless of the scope of application, most graphics APIs and GPU architectures use triangles as the basic primitive for constructing a scene. Polygonal models of complex shapes must be converted into a set of triangles before visualization. This process is called triangulation. The quality of the resulting triangulation affects not only the correctness of the geometric representation of the object, but also the rendering speed, the number of redundant rasterization operations, and the stability of calculations. Most known algorithms are focused on obtaining an acceptable partition in the minimum time, but do not take into account the efficiency of further scene display. Therefore, the task of developing a method for finding the optimal triangulation option from the standpoint of software performance is relevant. The relevance of the study also lies in the fact that the image of a realistically detailed scene requires large computing power.

The *object* of research in this article is the process of 3D visualization, the *subject* of research is triangulation algorithms. The study solves the *problem* of developing a triangulation method to optimize the process of 3D visualization, as well as implementing the method in the form of software for 3D visualization systems.

METHOD OF OPTIMAL TRIANGULATION OF POLYGONS

Let us define the problem of optimal polygon triangulation formally. Let us consider a simple polygon

$$P = \{v_1, v_2, \dots, v_n\},$$

where $v_i = (x_i, y_i)$ are polygon vertices.

It is necessary to find such a triangulation

$$T = \{t_1, t_2, \dots, t_{n-2}\},$$

or which the function is minimized

$$F(T) = \alpha A(T) + \beta D(T) + \gamma C(T)$$

where $A(T)$ is repainting area;

$D(T)$ is total deviation of triangles from equilateral shape;

$C(T)$ is computational complexity of processing;

α, β, γ are weighting factors.

The method involves generating admissible triangulation options followed by selecting the best solution based on the value of the function *F(T)*. Unlike standard algorithms, the proposed approach evaluates not one partition option, but a set of alternative configurations.

Thus, the main idea of the proposed method is to calculate all possible options for triangulating a polygon.

The initial action is to specify the vertex of the polygon, from which the triangulation process will begin. In the process of identifying all options for triangulating, it is necessary that each vertex of the polygon be the initial one. For convenience, the sides of the polygon are presented in the form of vectors that describe the polygon in a clockwise direction. The concepts of the starting, generating, previous, next and auxiliary vectors are introduced.

A *starting vector* is a vector from the beginning of which a vector is drawn that divides the polygon into two parts: a triangle and another part.

A *next vector* is a vector to the end of which a vector is drawn that divides the polygon into two parts: a triangle and another part. Also, the beginning of this vector is the endpoint of the starting vector.

A *generating vector* is a vector that has a common beginning with the starting vector and a common end with the next vector and that divides the polygon into two parts: a triangle and another part.

A *previous vector* is a vector that has an endpoint with the coordinates of the starting point of the starting vector.

An *auxiliary vector* is the inverse of the previous vector.

To construct the first triangle, the starting vertex is connected to the end of the next vector for the starting vector. There is a second method. The starting point is connected to the beginning of the previous vector for the previous vector for the starting vector. You can also use the third method, which is a combination of the first two.

Figure 1 shows the options for starting the polygon division.

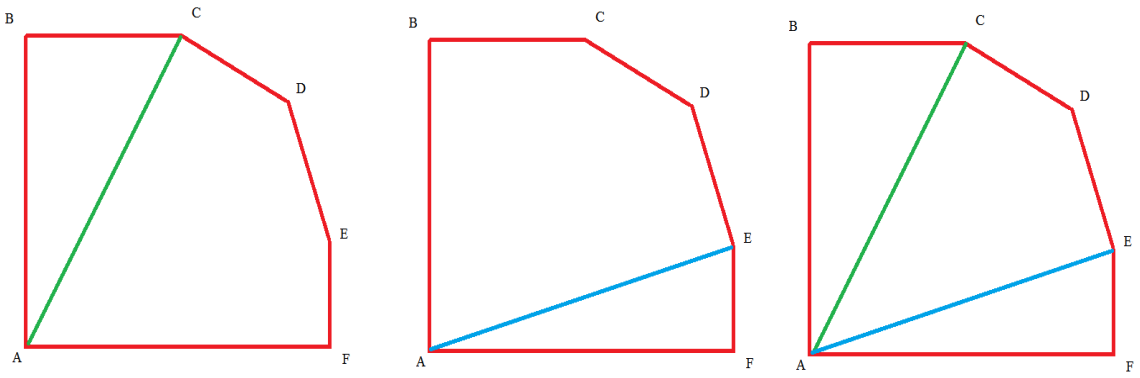

Fig. 1. Variants of starting polygon division

After that, the resulting triangle or triangles are cut off from the initial polygon, forming a new polygon. The triangulation procedure is performed recursively for the resulting polygon. The condition for exiting recursion is the situation when the newly created polygon is a triangle.

The algorithm also provides for checking the possibility of forming a triangle or triangles that will be cut off so that the generating vector does not intersect the other sides of the polygon (Figure 2) and does not supplement the polygon with a new part, generating it (Figure 3). In order for there to be no intersection of the generating vector with the sides of the polygon, it is necessary that no vertex of the polygon is in the middle of the resulting triangle for cutting.

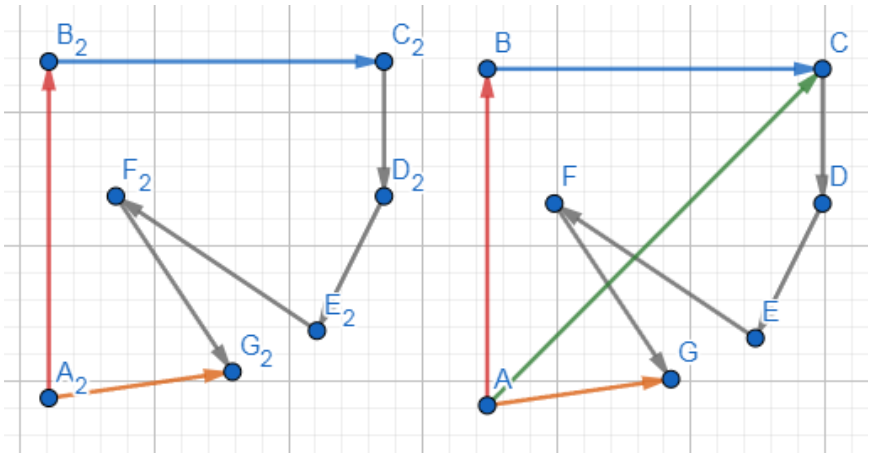

Fig. 2. On the left, it is the polygon before creating a triangle for clipping; on the right it is the polygon after creating a triangle for clipping

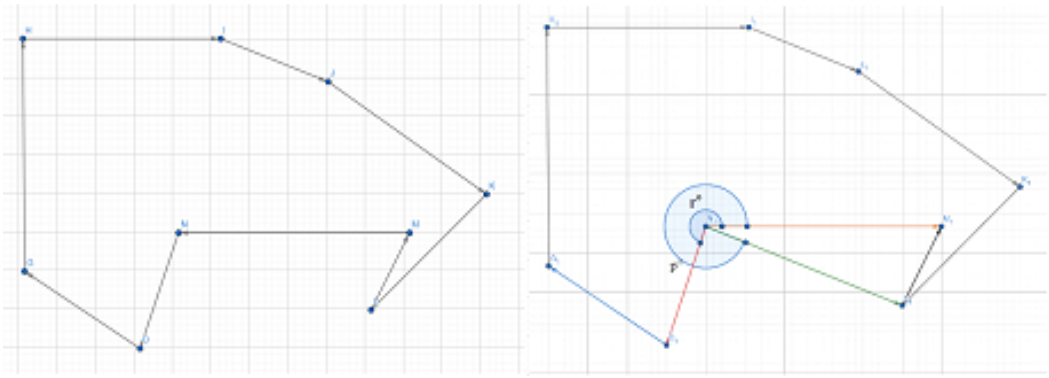

Fig. 3. Next step of the triangulation procedure

The check for whether a vertex is in this triangle is performed as follows. The sides of the triangle are represented as vectors in which the beginning of each subsequent vector is the end of the previous one. Each vector is chosen in turn as the starting vector, an auxiliary vector is constructed for it, and the condition is checked whether the counterclockwise angle between the auxiliary and starting vectors is smaller than that between the auxiliary and a vector formed with the same beginning as the starting vector and ending at the vertex being checked. If the condition is met for at least one starting vector, then the vertex does not lie in the middle of the newly created triangle.

Figure 4 shows the polygon formed after cutting off the initial triangle polygon using the first method.

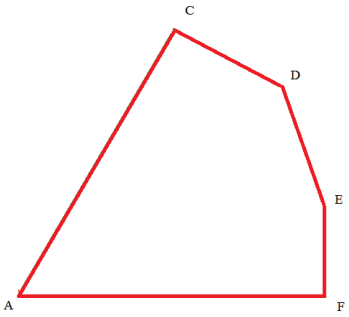

Fig. 4. Polygon formed after cutting from the original polygon

After obtaining the triangulation result, it is necessary to calculate the area that is drawn during prerendering, but will be discarded for the final rendering. This will be calculated by adding the area that will be discarded for each triangle. It is also necessary to determine how much each of the triangles into which the initial polygon was divided differs from a regular triangle. This will be implemented by calculating the sum of the differences of each of the angles of the triangles with sixty degrees.

Let's calculate the area of the rejected part D, as well as the deviation d for an arbitrary triangle other than a right triangle.

$$p = (a+b+c)/2 \tag{1}$$

$$S = \sqrt{p(p-a)(p-b)(p-c)} \tag{2}$$

$$x = max(x1, x2, x3) - min(x1, x2, x3) \tag{3}$$

$$y = max(y1, y2, y3) - min(y1, y2, y3) \tag{4}$$

$$D = xy - S \tag{5}$$

$$d = |60-q|+|60-r|+|60-n| \tag{6}$$

Since it is impractical to calculate all possible triangulation options in real time before displaying the polygon on the screen, the proposed method is intended for preliminary triangulation of polygons with determination of the optimal triangulation result.

Below are formulas that show how the amount of occupied space will change before and after the triangulation of an *n*-gon, where *n* is the number of sides, the number of vertices.

$$T = n-2 \tag{7}$$

$$V = 3(n-2) \tag{8}$$

$$M = 8V = 8(3(n-2)) = 24(n-2) \tag{9}$$

Now let's present this mathematical method at the algorithmic and software level.

GENERAL ALGORITHM AND SOFTWARE

The basis of the algorithmic support for optimal polygon triangulation is an algorithm with the following steps:

1. Obtain a list of polygon vertices.
2. Construct all admissible diagonals.
3. Recursively generate triangulation options.
4. Calculate criteria for each option.
5. Perform the optimized triangulation procedure according to the proposed method.
6. Select the triangulation with the minimum value.
7. Save the result in the model database or transfer to the renderer.

The following pseudocode corresponds to this: Listing 1.

Listing 1.

General algorithm for multi-criteria optimal triangulation

```
Input:
P = {v1, v2, ..., vn} // list of polygon vertices
α, β, γ // criteria weighting factors
Output:
Tbest // optimal triangulation
BEGIN
1. // Get a list of polygon vertices
READ(P)
2. // Construct all valid diagonals
D ← Ø
FOR i ← 1 TO n DO
FOR j ← i+1 TO n DO
IF IsDiagonalValid(vi, vj, P) THEN
D ← D ∪{(vi, vj)}
ENDIF
ENDFOR
ENDFOR
3. // Recursively generate all triangulation options
Variants ← GenerateTriangulations(P, D)
4. // For each option, calculate the criteria
BestScore ← +∞
Tbest ← NULL
FOR EACH T IN Variants DO
A ← ComputeOverdrawArea(T)
Q ← ComputeTriangleQuality(T)
C ← ComputeRenderComplexity(T)
5. // Perform optimized triangulation procedure
Score ← α*A + β*Q + γ*C
6. // Choose triangulation with minimum value
IF Score < BestScore THEN
BestScore ← Score
Tbest ← T
ENDIF
ENDFOR
```

Software has been developed to implement the proposed method. The architecture of the 3D visualization software system using the proposed method is shown in Fig. 5.

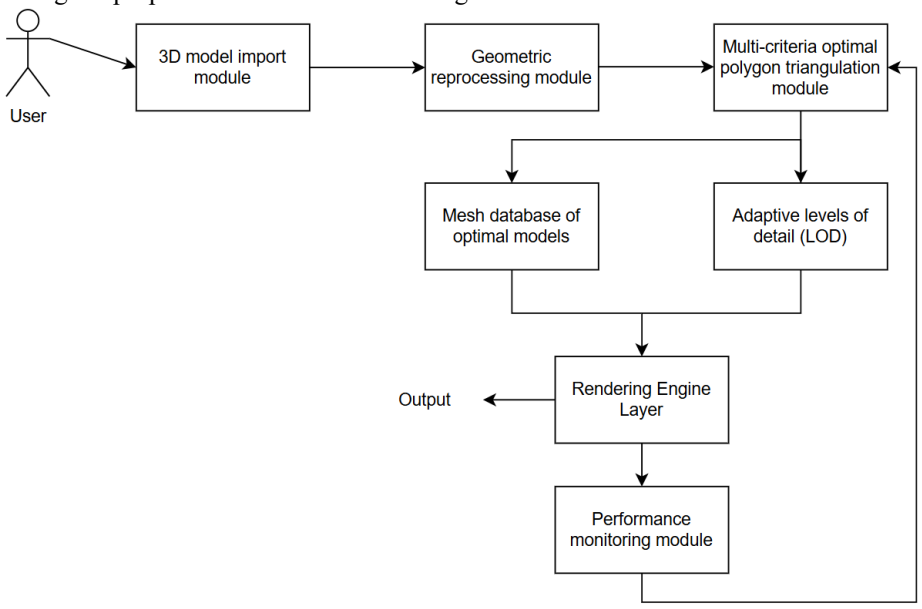

Fig. 5. Architecture of the 3D visualization software system

Let's consider the multicriteria optimal triangulation module in more detail.

The module input is pairs of numerical values, which represent the coordinates of the vertices of the polygon. An object of the class is formed from these pairs of numerical values *Vector*, which accepts axis coordinates into the constructor *x* and *y* starting point and axis coordinates *x* and *y* endpoint. The constructor also has optional parameters for references to the next and previous vectors, which represent the next and previous sides of the polygon, respectively. Each pair of values is interpreted as the coordinates of the start of the vector, and the next pair after it as the coordinates of the end point. For the last pair of values, the coordinates of the end point of the vector will be the first pair. Thus, an array of vectors is created. The next stage of the method is to build an order in the set of vectors. The corresponding function sets the values to the parameters *previous* and *next* class *Vector*, which are responsible for referring to the previous and next vector for a given class object *Vector* respectively, references to the previous and next objects in the array of class objects *Vector*. For the last object in the array, the parameter *next* – is a reference to the first object in the array, and the field *previous* for the first element in the array – this is a reference to the last element in the array. The resulting array of vectors is then passed to the class constructor *Polygon*, which sets the parameter *vectors* class *Polygon* this array.

Next, the resulting class object *Polygon* is passed as a parameter to the main function of the program, which is called *Tri*. First, the function checks whether the object represents the class *Polygon*, passed to it as a parameter, a triangle. To do this, it is checked whether the number of objects of the class is equal to *Vector* in the parameter *vectors*, passed to the class object *Polygon*, three. If so, the function returns an array with an object of the class *ChainPart*, the first parameter of whose constructor is an array of class objects *Polygon*, which display the triangles formed at this stage of triangulation, the second parameter is an array of class objects *ChainPart*, which display all possible options for dividing the polygon formed as a result of separation from the polygon displayed by the class object *Polygon*, passed to the function, that part that belongs to the triangles formed at this stage of triangulation. The third parameter of the constructor for this class is a string that reflects at which stage this object was created. This object of the class *ChainPart* The first parameter receives an array with the object passed to the function, the second parameter is an array without elements, and the third is the string "". If the number of objects in the vectors parameter of the object passed to the function is not equal to three, then the object is checked for correctness. The length of the parameter is checked *vector*s, the passed object is less than three. If not, then the object is correct. If yes, then the function returns an array with an object of class *ChainPart*, which has arrays without elements as both the first and second parameters in the constructor.

If the polygon represented by the object is not a triangle, but has more than three corners, then for each object of the class *Vector* in the parameter *vectors* An operation consisting of three stages is started. During the operation, the vector on which it is performed is called the "start vector".

Stage I.

In implemented in a separate function *firstPart* The first stage of the action set is displayed. First, a new object of the class is created *Vector* ("generator vector"), in the constructor of which the coordinates of the beginning of the "start vector" are passed as the coordinates of the endpoint and the coordinates of the end of the vector ("next vector"), to which the parameter refers *next* "start vector" as the coordinates of the starting point. Next, an object of the class is created *Polygon* ("new polygon"), whose constructor takes an array of three elements, which are copies of three objects: the newly created object "generator vector", "start vector" and "next vector". After that, another object of the class is created *Polygon*, whose constructor accepts an array, the first element of which is the inverse of the "generator vector", and the next elements are all other objects of the parameter *vectors,* passed to the function object, in the same sequence without the "start vector" and "next vector". This object is called "new polygon".

Function *firstPart* returns an array of two elements: an object of class *ChainPart*, whose constructor parameters are an array with the "polygon" object and the result of the function *Tri* above the object "new polygon" and the line «*firstPart*» and an object that is a copy of the "generator vector".

Stage II.

In implemented in a separate function s*econdPart* The second stage of the action set is displayed. First, a new object of the class is created *Vector*, whose constructor is passed the coordinates of the beginning of the "start vector" as the coordinates of the starting point and the coordinates of the beginning of the "pre-preceding vector" referred to by the parameter *previous* "previous vector" to which the parameter refers *previous* "start vector" as the coordinates of the starting point. Then the same actions are performed as in the first stage, but instead of "start" and "next vectors" now "previous" and "before previous vectors".

During the execution of each of the first two stages, after creating the "polygon" object, a check is made to see if there is any vertex of the polygon inside it, from which the triangle displayed by the "polygon" object is "separated". For each element in the array *Vector*s object "polygon" is taken as a parameter *previous*, then an object ("auxiliary vector") is created that displays the inverse vector for the vector displayed by the parameter *previous*. At the same time, the element itself at this time has the name “starting vector”. Then the angle against the time arrow between the “auxiliary vector” and the “starting vector” is searched for. Next, an object (“new vector”) of the class is created *Vector*, which displays a vector formed with the beginning at the starting point of the "start vector" and the

end at the vertex of the polygon, which is checked to see if it lies in the middle of this triangle. The function that calculates the counterclockwise angle between the vectors uses the functionality of the library *numpy/*.

Stage III.

If both the first and second stages were performed in the function, then the function *thirdPart* the third stage is performed. This stage combines the actions of the first and second stages in terms of separating two triangles obtained in the previous two stages.

Function *thirdPart* returns an object of the class *ChainPart*, whose constructor parameters are an array with the "polygon" object and the result of the function *Tri* above the object "new polygon", and the line «*thirdPart*».

RESULTS AND DISCUSSION

Table 1 shows the overall deviation (Difference) triangles from a right triangle in degrees and area (Area) of pixels, which will be repainted during the rendering process in conventional units in increasing area for all triangulation options for the test polygon.

It can be seen that the most advantageous (smallest) area differs from the least advantageous (largest) by almost three times, which means that the time required to repaint pixels can be significantly reduced by using the first option.

Figure 6 shows a diagram showing the correspondence of the parameters "Area of the part to be repainted" (Area) and "The degree of difference from a right triangle" (Difference) for different test polygon triangulation results.

Table 1

Results of the parameters of the obtained triangulation options

Difference	Area
362,52040941662386	53,0
429,39030706246797	56,0
558,7806141249359	57,00000000000001
625,6505117707799	60,0
497,59482141998217	61,0
...	...
712,6198649480405	108,0
631,5218586636224	111,00000000000003
698,3917563094665	114,00000000000003
840,0000000000001	116,0
832,6198649480402	121,0
912,5852594569593	128,00000000000006
960,0	129,0
1032,5852594569594	141,00000000000006
904,6948869988842	144,00000000000003
1024,6948869988842	157,00000000000003

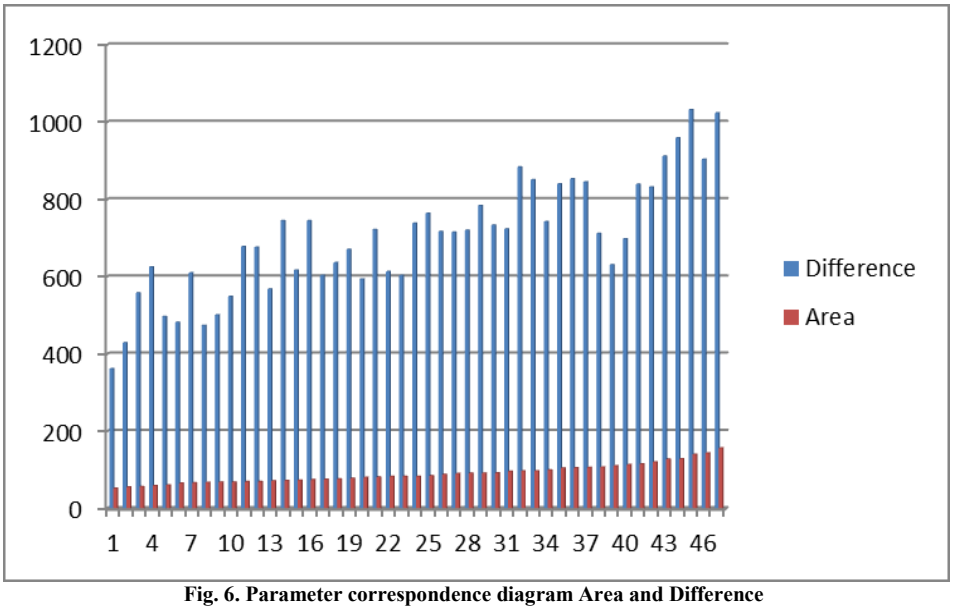

Fig. 6. Parameter correspondence diagram Area and Difference

It can be seen from it that there is not always a direct relationship between these parameters.

CONCLUSION

The article solves the current scientific and applied problem of increasing the efficiency of 3D visualization software systems by improving the process of triangulation of polygonal models. The analysis of modern research has shown that most existing methods are focused mainly on the speed of constructing a triangular mesh or on improving its geometric quality, but do not sufficiently take into account the influence of the triangulation structure on the subsequent rendering performance.

A method of multi-criteria optimal triangulation of polygons is proposed, which is based on the formation of a set of permissible partition options and the selection of the best solution according to a set of criteria: minimizing the area of re-painting of pixels, reducing the deviation of triangles from the rational form and reducing computational costs during visualization. Unlike traditional algorithms, the proposed approach is focused not only on the correctness of the geometric representation, but also on the efficiency of subsequent graphic processing.

Algorithmic and software implementation of the method has been developed, which includes the construction of admissible diagonals, recursive formation of alternative triangulations, multi-criteria evaluation of results and selection of the optimal configuration. The architecture of a 3D visualization software system is proposed, in which the optimal triangulation module is integrated with the geometry processing, model saving and rendering subsystems.

The results of the experimental study confirmed the effectiveness of the proposed approach: it was established that the best triangulation option for the test polygon provides an almost threefold reduction in the area of repeated repainting compared to the worst option, which indicates the possibility of reducing the visualization time and increasing the system's productivity. It is also shown that there is no direct relationship between the geometric correctness of triangles and the area of repainting, which confirms the feasibility of using the multi-criteria approach.

The practical significance of the results obtained lies in the possibility of applying the method in graphics engines, CAD/CAM systems, simulators, VR/AR applications, digital twins and other software complexes where speed and quality of display of complex scenes are important. Prospects for further research lie in reducing the time to find the optimal solution through heuristic strategies and dynamic programming, implementing a GPU-accelerated version of the algorithm, and using machine learning methods to predict the optimal triangulation structure for different types of polygonal models.

References

1. Long Chen, Jin-chao Xu. Optimal Delaunay Triangulations. Journal of Computational Mathematics. Vol. 22, No. 2, 2004, pp. 299-308.
2. Zhang Z., Wang Y., Jimack P. K., Wang H. MeshingNet: A New Mesh Generation Method based on Deep Learning. arXiv preprint arXiv:2004.07016, 2020. 13 p.
3. Chen X., Liu J., Yan J., Wang Z., Gong C. An Improved Structured Mesh Generation Method Based on Physics-informed Neural Networks. arXiv:2210.09546, 2022. 14 p.
4. Siddiqui Y., Alliegro A., Artemov A., et al. MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers. arXiv:2311.15475, 2023. 16 p.
5. Elshakhs Y. S. A Comprehensive Survey on Delaunay Triangulation. Cranfield University Repository, 2024. 60+ p.
6. Curved Image Triangulation Based on Differentiable Optimization. Computer Graphics Forum, 2024.
7. Optimization of a Triangular Delaunay Mesh Generator using Reinforcement Learning. Computer-Aided Design, Vol. 180, 2025, Article 103964.
8. Xiao J. et al. PI-MeshONet: A Generalizable and Self-supervised Method for Mesh Generation. Knowledge-Based Systems, 2025.
9. Direct Rendering of Intrinsic Triangulations. ACM Transactions on Graphics, 2025. DOI:10.1145/3716314.
10. Lei N., Li Z., Xu Z., Li Y., Gu X. What's the Situation with Intelligent Mesh Generation: A Survey and Perspectives. IEEE Access / arXiv:2211.06009, 2022. 29 p.