Method for finding and storing optimal triangulations based on square matrix

M. Saračević, S. Mašović, P. Stanimirović, P. Krtolica

Abstract. The process of finding an optimal triangulation is a procedure in computational geometry and computer science requiring an amount of time for execution as well as relatively large requirements for data storage. Therefore, it is important to provide an efficient way for faster generation of optimal triangulations and their storage in as little memory space as possible. Accordingly, the paper presents a new method for finding an optimal triangulation based on the square matrix (SM) method, which stores all results of triangulations for given n. The main emphasis of the method is on the speed of generating optimal triangulation and saving memory space during calculation of a large number of triangulations. The method of data storage is constructed on the idea that it is not necessary to store each weight of triangulation particularly, but performing it all at once. The implementation in Java environment is described and experimental results of the SM method are compared to the Hurtado-Noy method.

M.S.C. 2010: 68U05, 32B25.
Key words: Computational Geometry; optimal triangulations; square matrix; minimum weight triangulation; Java programming.

1 Introduction

Polygon triangulation is the process of decomposition a certain area to triangles with non-intersecting sides. The points that shape this area in the same time are the vertices of at least one of the triangles. Polygons can be used in approximations and graphical representation of curves.

The procedure of triangulation is done in $2D$, $3D$ or multi-dimensional space and is seen as process of finding systematically modeled triangles. A triangulation in $2D$ is realized using the coordinates (x, y) of the polygon vertices. The most important characteristic of one triangulation algorithm is its uniqueness, its independence from starting point and from the order of operations in the implementation. The quality of a triangulation algorithm is measured by the number of performed operations and by the storage of obtained results in computer memory.

Polygonal triangulation as an important procedure applicable in the computer graphics represent a basic and preprocessing steps for most nontrivial operations with the polygons [4]. Triangulation of a convex polygon is an important problem that appears in a $2D$ computational geometry [11],[13]. The decompositions of $2D$ scenes are used in contour filling, clipping, cast shadow removal and convex hull fitting. Triangulation techniques are the most common methods in computational geometry. The easiest way of segmentation and smoothing of double curved area is construct a method from network of triangles who are contained in them.

The advantage of the triangle as a geometric object is that the area between the three points is always flat. Other advantages of triangulation methods are small deviations from the original form, good structural properties and the possibility of overlaying complex free forms. This procedure is crucial for the speed, quality and resolution of the $3D$ objects and pre-trial phase of non-trivial operations of simple polygons [6].

The polygon triangulation is the technique used in modelling of $3D$ objects. Tools for rendering either generate $3D$ models or load them when creating the desired images. Apart from the geometry of the models lighting, shadows, textures, colors, transparency and some advanced features of individual software packages are also today tools for rendering in the output file implanted are also.

The process of finding the triangulation is based on the Minimum-Weight Triangulation (MWT) algorithm. Due to difficulty of finding the exact solutions of MWT, many authors have studied heuristics, which in some cases may find the solutions although they cannot be proven they work in all cases. Most of the research are focused on the problem of finding sets of edges that are guaranteed the minimum-weight triangulation.

2 Related works

There are many researches that treat optimal triangulations. In the paper of the Mulzer and Rote [15] the MWT problem is observed as the problem of finding a triangulation from the set of given points that minimizes the sum of the edge lengths. In [5] Eppstein shows that the length of MWT of a point set can be approximated within a constant factor. In $O(n \log_n)$ time they compute a triangulation with $O(n)$ new points, and no obtuse triangles, that approximate the MWT.

In [10], the authors presented algorithms that construct MaxMin and MinMax area triangulations of a convex polygons in $O(n^2 \log_n)$ time and $O(n^2)$ space. The algorithms in [16, 9] use dynamic programming and a number of geometric properties. Mirzoev and Vassilev [14] consider the problems of finding two optimal triangulations of convex polygon: MaxMin area and MinMax area. These are the triangulations that maximize the area of the smallest area triangle in the triangulation, and respectively minimize the area of the largest area triangle in the triangulation, over all possible triangulations. In [1, 19] are given linear program for minimum-weight triangulation and method for minimum weight pseudo-triangulations. In comparison to the previous researches in this field, we propose a new method for efficient storing and finding an optimal triangulation.

3 The method for optimal triangulation

The initial idea behind the implementation of the constructed method for storing weights resulted from advanced Java capabilities that are related to application of packages for working with table cells. The `DefaultTableCellRendererJava` package is taken the main role in storing weight triangulations. This package enables storage of multiple values within a single table cell (which provide efficient cell division into multiple columns and rows). The calculation of MWT is based on dynamic programming techniques. Apart from fast calculation of optimal triangulations, one of the goals in the implementation is to save memory space during calculation of a large number of triangulations.

The method is constructed on base of different ways of parenthesizing matrix multiplications (optimal scheduling of matrix multiplication) and the associative law $A \cdot (B \cdot C) = (A \cdot B) \cdot C$ for multiplication of matrices.

Let assume that we have a sequence $A_1, A_2, \ldots, A_n$ of n matrices to be multiplied and let $P(n)$ is the number of different parenthesizing of this product. This product of n matrices we can split between kth and $(k+1)$th matrices for any $k = 1, 2, \ldots, n-1$ and parenthesize the two resulting sequences independently as $((A_1 A_2, \ldots, A_k)(A_{k+1} A_{k+2}, \ldots, A_n))$. the recurrence (3.1) is obtained in this way:

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_k^{n-1} P(k)\, P(n-k), & n \geq 2. \end{cases} \tag{3.1}$$

This recurrence generate the sequence of Catalan numbers and satisfies $P(n) = C_{n-1}$. The optimal triangulation of convex polygon is constructed as dynamic programming problem and consist the following steps:

1. labeling of edges with matrix $A_i, i = 1, 2, \ldots, n$ $(A_{n\times 1})$ odd edges and $A_{1\times n}$ even edges;
2. splitting the problem into subproblems using optimal split;
3. parenthesize the subproblems optimally;
4. combination of optimal subproblem solutions.

Let us denote the weight of a minimal triangulation convex polygon as $m[1, n]$. The weight of each triangle in triangulation is the length of its perimeter and let denote $w(i, j, k)$ the length of perimeter of $\Delta v_i v_j v_k$.

The convex polygon contains sub-polygons with i vertices through k $(2 \leq k \leq i)$, and we denote the weight of a minimal triangulation of such a sub-polygon as $m[i, j]$ (is the minimum number of scalar multiplications needed to compute $A_{i\ldots j} = A_i \cdot A_{i+1} \cdots A_j$. Possible values of $m[i, k]$ fall into two cases.

- **Case 1**: if $i = j$ then $m[i, j] = m[i, i] = 0$, $A_i = i$ and multiplication of matrices is not need.
- **Case 2**: if $i < j$, then we assume optimal split at k, $i \geq k < j$. In this case $A_{i,k}$ and $A_{k+1,j}$ have a dimension $v_{i-1} \times v_k$, $v_k \times v_j$ respectively. For each choice of j we calculate

$$m[i,j] = m[i,k] + m[k+1,j] + w(i,j,k) \quad for \quad k = i, i+1, i+2, \ldots, j-1. \tag{3.2}$$

These two cases give the following recursive formula (3.3):

$$m\,[i,\ j] = \begin{cases} 0 & if\ i = j \\ \min_{i \le k < j} m\,[i,\ k] +\ m\,[k+1,\ j] +\ w(i,j,k) & if\ i < j \end{cases} \tag{3.3}$$

The convex polygon $v_0 v_1 \ldots v_{n-1}$ with $(n-3)$ non-intersect diagonals is divided into $(n-2)$-triangles. In most cases, it is possible to find a criterion for calculating weight of triangulation, i.e., a values of the perimeter, sum of heights or length of the longest median by the number of triangles [14]. The optimal triangulation can be calculated with the recurrent formula (3.3). In sequel we describe the algorithm for computing the weight (w) of the triangle $w(i,j,k)$.

For the illustration purpose, observe two triangulations of convex heptagon, as shown in Figure 1.

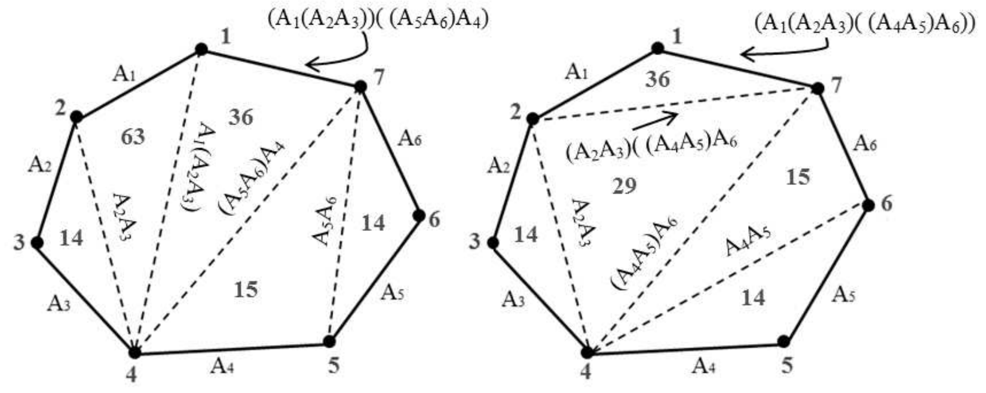

Figure 1: Two triangulation of convex heptagon with associated weights

3.1 Storing weights based on a square matrix (SM model)

The method for storing triangulation weights is based on a square matrix $(i = j = n)$ with dimensions $(n \times n)$, where i stands for number of rows, j represents columns, and n is the number of polygon vertices. The matrix $\mathcal{A}_{ij}$ is used for organization of k-vertices of the triangle (i,j,k) and presentation of their weights (w).

In our particular case, the matrix A can be permanently transferred as a table by using `Java JDBC API`. The matrix is diagonally divided into two parts. The diagonal divides $i = j$ the matrix into left and the right part. Positions $(1,1), (2,2), (3,3), \ldots, (n,n)$ are filled with zero (see Figure 2).

The k-vertices of the triangles are recorded on the left side of the matrix while their calculated weights (3.3) are on the right side:

$$k[i,j] = w[j,i]$$

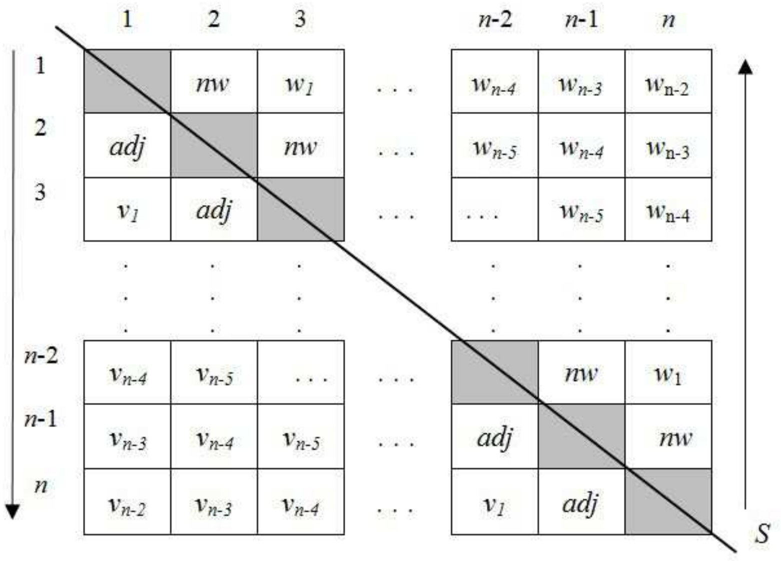

Figure 2: The general scheme of filling the matrix

Example 3.1. This example shows how many k values there are in the set v_k for pentagon (Figure 3).

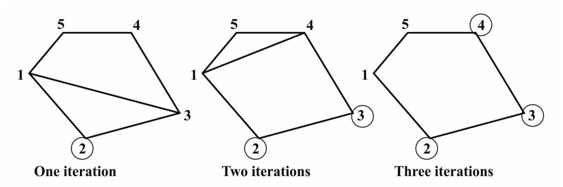

Figure 3: An example of increasing the number of k-vertex in the field of square matrix

There is no vertex k between vertices 1 and 2, because they are adjacent (labeled with adj) so that there is no weight on the other side of the diagonal (labeled with nw, Figure 2).

More precisely, there is no (i, j, k) triangle that has defined weight. The number of vertices between (i, j) increases as the number of i-rows increases. For example, there is one vertex $k = 2$ between vertices 1 and 3, there are two vertices $k = 2, 3$ between vertices 1 and 4, there are three vertices $k = 2, 3, 4$ between vertices 1 and 5, and etc.

As the distance between vertices (i, j) increases, so does the number of possible k-vertices within the set v and the number of weights (w) on the other side of the diagonal (j, i).

Based on $P(n) = C_{n-1}$ for $n = 5$ (pentagon) we obtain five different triangulations. There is only one universal matrix (5×5) for all five triangulations containing all triangle weights necessary to acquire minimal weight for each triangulation separately.

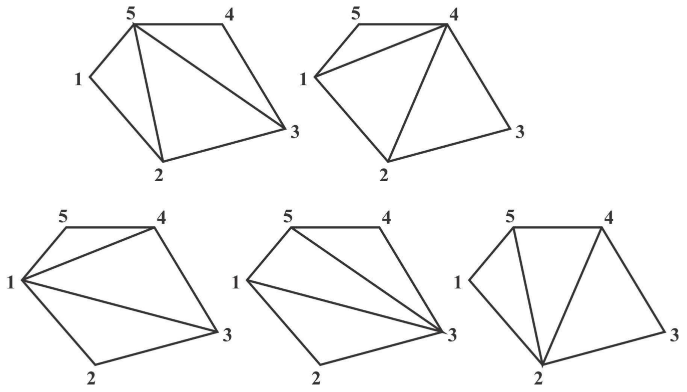

Figure 4: Pentagon triangulations

In the following we presents a 5×5 matrix for pentagon according to the filling general scheme.

$$opt5 = \begin{bmatrix} 0 & \{nw\} & \{25\} & \{25,30\} & \{16,80,35,60\} \\ \{adj\} & 0 & \{nw\} & \{10,30\} & \{35,11,15\} \\ \{2\} & \{adj\} & 0 & \{nw\} & \{30\} \\ \{2,3\} & \{3\} & \{adj\} & 0 & \{nw\} \\ \{2,3,4\} & \{3,4\} & \{4\} & \{adj\} & 0 \end{bmatrix}$$

The method of storing weights is combined with our method for generating triangulations [18].

We present Algorithm 3.1 for storing triangulation weights and finding optimal triangulation. At the beginning, the algorithm expects number of polygon vertices n, where each vertex gets its own coordinates. The table gets expanded with a new column that store triangulation weights of the current process.

Algorithm 3.1 contains three main steps:

1. Form a square matrix $i \times j$ and we fill the fields along the diagonal with 0 (where $i = j$). Then we fill with *adj* on the position on the position (i, j) and by *nw* on position (j, i) with respect the first diagonal line (see Figure 2). These are adjacent vertices and there are no k-values for them, which can be used to form (i, j, k) triangle. Those fields have value 0 and after that we add all possible values for k between (i, j).

2. Calculate all triangulation weights (w) for rows and columns of the matrix where $(j - i) > 1$ according to the formula (3.1) and proceed to the next step. We add corresponding weights to assigned k values on the position (j, i).

3. The obtained weights are assigned to the corresponding polygon triangulation.

Algorithm 3.1 Algorithm for storing weights and finding optimal triangulation.

```
Require: $n$, table $T_n$
1: Create a new matrix $(i \times j)$
      for $(i = 1; i \leq n; i++)$
            $v[i, j] = 0$, where is $i = j$
            $v[i, j] = adj$, where is $j - i = 1$
2: Filling the other fields
      for$(i = 1; i \leq n; i++)$
      for$(j = 2; j \leq n; j++)$
         $j = i + 1$
            for$(k = i + 1; i \leq j - 1; k++)$
               $m[i, j] = min\{m[i, k] + m[k + 1, j] + w(i, j, k)\}$
               $W[T_i] = sumAll[m[i, j]]$
                  if $W[T_i] < m[i, j]$ then
                  $W[T_i] = m[i, j]$
                  return $m[i, j]$
3: Selection $(n - 2)$ of triangles $(i, j, k)$ from the matrix and identification in $T_n$
      Evaluate $Opt[w] = minW[T_i]$
```

Example 3.2. Obtaining optimal triangulation for the irregular convex pentagon is explained in four steps.

Step 1:Form a 5×5 matrix, and diagonally fill out fields with zeroes and values for adjacent vertices. Afterwards, we fill out values for all (i, j, k) triangles. After these steps, the matrix has the following form:

$$opt5 = \begin{bmatrix} 0 & \{nw\} & \{35\} & \{35\} & \{16\} \\ \{adj\} & 0 & \{nw\} & \{10\} & \{11\} \\ \{2\} & \{adj\} & 0 & \{nw\} & \{30\} \\ \{\langle 2\rangle, 3\} & \{\langle 3\rangle\} & \{adj\} & 0 & \{nw\} \\ \{2, 3, \langle 4\rangle\} & \{3, 4\} & \{4\} & \{adj\} & 0 \end{bmatrix}$$

Step 2: Find the sums of weights (w) for C_{n-2} rows given on the table T_5. The table was created by using algorithm for generating triangulations, which is described in detail in [17].

The table for storing triangulations gets expanded by a new column W (Table 1).

Table 1: Extended table $\mathcal{T}_5$

i	D_1	D_2	**W**
1	$\delta_{1,3}$	$\delta_{1,4}$	76
2	$\delta_{1,3}$	$\delta_{3,5}$	71
3	$\delta_{2,4}$	$\delta_{1,4}$	61
4	$\delta_{2,4}$	$\delta_{2,5}$	37
5	$\delta_{2,5}$	$\delta_{3,5}$	51

The sums of all triangle weights are saved in the new column W for $[T_i]$, where i is the current row.

The diagonals in the rows form $(n-2)$- triangles and for each triangle, we take the weights from the matrix A:

$$\begin{aligned} W[T_1] &= 25+35+16=76 \\ W[T_2] &= 16+25+30=71 \\ W[T_3] &= 10+35+16=61 \\ &\ldots \end{aligned}$$

Step 3: In this step we calculate the lowest sum in a row (optimal triangulation):

$$OptT_w = min\{W[T_1],\ldots,W[T_i]\} \tag{3.4}$$

where is $1 \leq i \leq T_n$.

Based on (3.4) we get optimal triangulation $OptT_w = 37$. After finding the optimal triangulation from previous table, it follows drawing the same one on the basis of internal diagonals $\delta_{i,j}$ that correspond to the row and column in the matrix, i.e. pair (i,j): $optT = \{\delta_{2,4}, \delta_{2,5}\}$ (Figure 5).

$$\begin{array}{c|ccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \hline \mathbf{1} & 0 & \{nw\} & \{25\} & \{35\} & \{16\} \\ \mathbf{2} & \{adj\} & 0 & \{nw\} & \{10\} & \{11\} \\ \mathbf{3} & \{2\} & \{adj\} & & & \\ \mathbf{4} & \{[\mathbf{2}],3\} & \{[\mathbf{3}]\} & & & \\ \mathbf{5} & \{2,3,[\mathbf{4}]\} & \{3,4\} & & & \end{array}$$

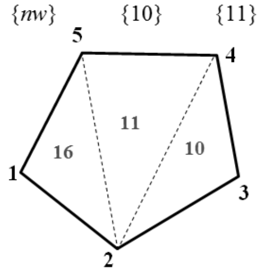

Figure 5: The corresponding values in the matrix and optimal triangulation

3.2 Comparative analysis

Now let us look on savings, in terms the calculations for finding the weight of each triangulation separately when it comes with the Hurtado-Noy method.

In the worst case, the total number of weights calculations (with repetition) is the product of the total number of triangulations of the n-gon and the number of its diagonals:

$$TM_w = T_n \times (n-2) \tag{3.5}$$

If we want to calculate triangulation weights of the pentagon, based on (3.5), we have 15 calculations in total (5 different triangulations with 3 triangles each).

Now we have to find savings in terms of calculating triangle weights (i, j, k). At the beginning we determine how many k-values there are in the matrix. According to the general scheme of filling the matrix (Figure 2), the number of k-vertices increases as we go further from the diagonal. The first row does not contain elements (since they represent the so-called adjacent diagonals), the second has one element, the third has two and so on. The last diagonal row contains $(n-2)$ k- vertices since the difference between the first and the last n-vertex is always two, because the first and the last vertex are always subtracted from the total (Figure 6).

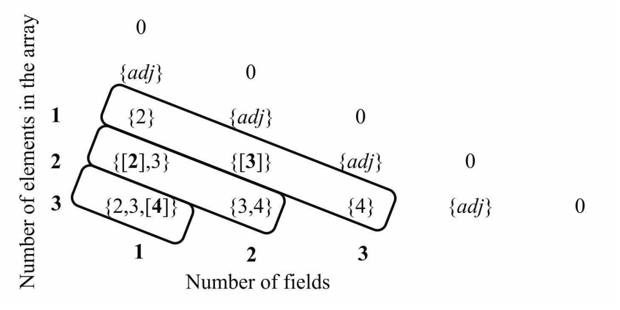

Figure 6: Example of calculating the total number of k-vertex

If we label the total number of stored k-vertices for a polygon with n-vertices with V_n then the number of (i, j, k) triangles or k-vertices that have been inserted into the table, are obtained from the following equation:

$$\begin{aligned} V_n &= \sum_{i=1}^{n-2} i((n-2)-(i-1)) \\ V_n &= \sum_{i=1}^{n-2} i(n-i-1) \end{aligned} \tag{3.6}$$

By decomposing formula 3.6 we can present the value of V_n as the sum of number of fields in rows and number of sequenced elements of that field.

Example 3.3. In this example we calculate the number of k-vertex for $n = \{5, 6, 7\}$, which are marked with V_5, V_6 i V_7:

$$V_5 = (1 \times 3) + (2 \times 2) + (3 \times 1) = 10 \quad \text{(filled out a total of 10 k-values)}$$
$$V_6 = (1 \times 4) + (2 \times 3) + (3 \times 2) + (4 \times 1) = 20$$
$$V_7 = (1 \times 5) + (2 \times 4) + (3 \times 3) + (4 \times 2) + (5 \times 1) = 35$$

For triangulations of a pentagon (Figure 4), there are 10 diagonals including repetition (5 different triangulations containing 2 diagonals each), shown in the form as follows: $\delta_{2,5}\delta_{3,5}, \delta_{2,4}\delta_{1,4}, \delta_{1,3}\delta_{1,4}, \delta_{1,3}\delta_{3,5}$ and $\delta_{2,4}, \delta_{2,5}$. Based on (3.6) it follows that the number of calculated weights is equal to the number of triangles:

$$SM_w = \sum_{i=1}^{n-2} i(n-i-1) \tag{3.7}$$

Therefore, we have 10 different triangles in the triangulation for a pentagon presented in Example 3.3. Using the traditional calculation for all triangulations of a pentagon separately i.e. by using $T_n \times (n-2)$ we get 15 triangles (including repetition) for all triangulations of a pentagon. According to $TM_w - SM_w$ we have saved 5 calculations.

For the second comparison, with our SM method, we will take a well-known algorithm given in the paper [8]. Hurtardo and Noy proposed an algorithm for generating the triangulations of P_n based on the triangulations of P_{n-1}. Moreover, they defined the tree of triangulations where all triangulations of P_n, i.e., the triangulations from $\mathcal{T}_n$, are arranged at the level n of this tree (based on this tree, can be calculated number of the weights in the storage).

Every triangulation at the level n has a "father" in $\mathcal{T}_{n-1}$ and two or more "sons" in $\mathcal{T}_{n+1}$. The sons of the same father are "brothers". There is an ordering among the children of a triangulation, and consequently among all triangulations. In the case of Hurtado-Noy hierarchy (or algorithm), we have the following. For every triangulation at level $n-1$ we need to perform $2n-5$ checks to find the diagonals incident to the vertex $n-1$. Total number of these checks is $(2n-5)C_{n-3}$.

Further, we must go through diagonals and copy some without transforming, while some of them should be transformed and two new diagonals should be inserted, for every incident diagonal which has been found. In such a way we make $2n-3$ pairs describing one new triangulation. The total number of incident diagonals is equal to C_{n-2}. All together, in the case of Hurtado-Noy algorithm we need

$$HN_w = (2n-5)C_{n-3} + (2n-3)C_{n-2}$$

number of weights calculations.

As the number of vertices increases, the number of repetitions of calculations drastically increases as well (also, speedup and savings are drastically increased, see Table 2).

Table 2: Comparative analysis

n	TM_w	HN_w	SM_w	D	R
7	210	45	35	10	**1.29**
8	792	161	56	105	**2.88**
9	3003	588	84	504	**7.00**
10	11440	2178	120	2058	**18.15**
11	43758	8151	165	7986	**49.40**
12	167960	30745	220	30525	**139.75**
13	646646	116688	286	116402	**408.00**
14	2496144	445094	364	444730	**1222.79**

TM_w – Total number of weights calculations (with repetition) ($T_n \times (n-2)$)
HN_w – Number of calculations based on Hurtado-Noy hierarchy
SM_w – Number of calculations based on Square matrix method (based on (3.7))
D – Difference of HN_w and SM_w (**savings**, SM-HN)
R – Ratio of HN_w and SM_w (**speedup**, SM/HN)

The method developed in this research is avoiding the same calculations and give rule for the calculation of the identical (i, j, k) triangles. We get the significant increase of the same calculation using the traditional method is noticed when it comes to $n > 7$ versus SM method.

4 Implementation and experimental results

Java NetBeans environment [7] has the package `OptimalTriangulation` that works as a supplement to the application that generates triangulations of a convex polygon. The main class operates through `compute()`, and the method calculate all weights within the table T_n.

In order to work with table cells, we have used `DefaultTableCellRendererclass` of the `Swing` package. This class inherits the class `Table` and allows manipulation of table cells (in this case, it allows us to assign a number of independent values to a single table cell).

Application of `Geometry` package and `GeometryInfo` class [2],[3] in the process of combining methods for finding the optimal triangulation and generating triangulations is shown in in Figure 2. The `GeometryInfo` class with its method `TRIANGLE_ARRAY` encompasses a set of three vertices that form triangle (i, j, k) and by using `GL_TRIANGLES` method it forms triangles.

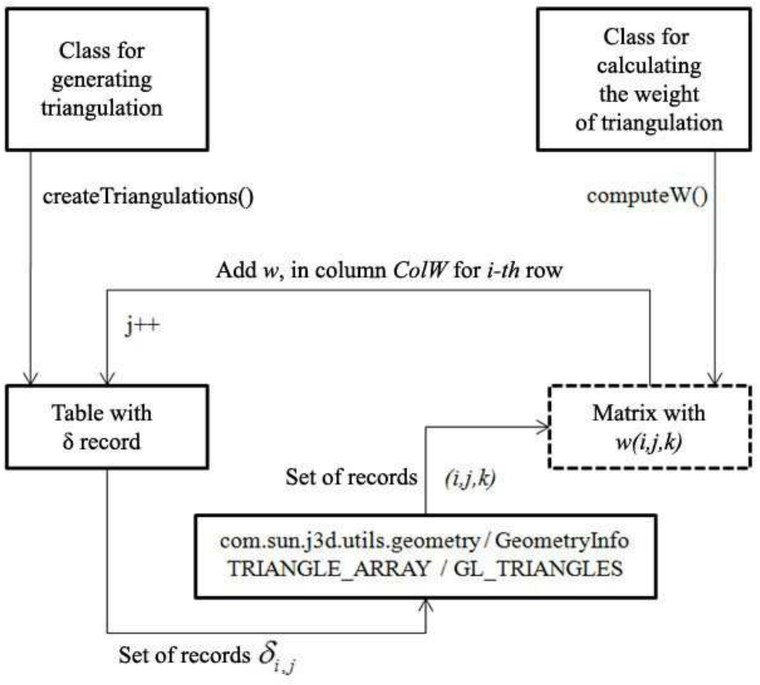
Class for generating triangulation
Class for calculating the weight of triangulation
createTriangulations()
computeW()
Add w, in column ColW for i-th row
j++
Table with δ record
Matrix with w(i,j,k)
Set of records
(i,j,k)
com.sun.j3d.utils.geometry / GeometryInfo
TRIANGLE_ARRAY / GL_TRIANGLES
Set of records $\delta_{i,j}$

Figure 7: The procedure for storage of triangulation

For the purpose of comparing experimental results with the SM method we present application implemented in Java environment. A table of stored weights corresponding to all possible triangulations of a certain polygon is obtained as a result of the application.

The application give a option of the graphic representation of optimal triangulation, and is performs as is described in Algorithm 3.1:

1. The weights are calculated by pressing the button *CALCULATE WEIGHTS*.
2. Upon successful calculation, checking of JDBC connection follows.
3. When the message for successful connection is displayed, corresponding table T_n that has been formed using method for generating all triangulations is called upon. Finding table T_n is simple (number n gets the same value as the number of vertices drawn in *JPanelapplication*).
4. Upon detecting adequate table T_n (if it exists), new columns for weight (w) will be filled out one by one for each triangulation (i.e. row by row). Filling out a new column means detecting (i, j, k) matrix values based on diagonals ($\delta_{i,j}$) that are already in the table T_n.
5. After all weights for triangulations are calculated, a user will get the output message status-OK for each triangulation.
6. By clicking *Show Optimal Triangulation* the JPanel displays the optimal triangulation.

The importance of storing in the proposed way reflects in the fact that the results of using this method can in an efficient way display optimal triangulation under the condition that there is a method of generating all triangulations.

Table 3 presents the results for n = 5,6,...,14. Table shows time needed to calculate weights (*step 1 and 2, without graphically generating optimal triangulation*) and the total time for finding optimal triangulation (*step 3 and 4, with display*).

Table 3: Experimental results in Java software solution (Time expressed in seconds)

n	Number of triangulation	Time to fill the matrix	Total execution time
5	5	0.01	1.7
6	14	0.03	2.4
7	42	0.09	2.8
8	132	0.12	3.7
9	429	0.19	5.2
10	1430	0.27	14.8
11	4,862	0.41	27.4
12	16,796	0.62	48.7
13	58,786	1.04	64.6
14	208,012	1.59	85.5

The testing was done on a computer with following technical specification: *CPU Intel Core2 Duo, 2.40GHz,Cache 4MB, RAM: 2Gb, Graphic: NVIDIA GeForce 8600M GS.*

5 Conclusion

The method for storing is based on the idea that during the process of calculation all data are presented in the form of a square matrix which is valid for all triangulations of a given value n.

The final goal of this method is to avoid recomputation of weights of already generated triangulations,and compute the weight of arbitrary triangulation only once. Also, all computed weights are stored into a single matrix which is exploited to prevent recomputations of weights. The triangulation corresponding to the minimal weight is generated using data stored in that matrix.

The method for storing and finding optimal triangulations represents an expanded version of a method for generating triangulations that has been implemented in `JavaNetBeans` environment. Thus, this method for storing weight triangulations is initiated by Java services for working with databases.

The significance of the constructed method is reflected in the fact that using the recorded values can be a very effective way to find the optimal triangulation. The obtained values from these techniques give a good direction for drawing of optimal triangulation in highly effective way.

The advantages of this type of storing temporary results are multiple, here are some of them: (1) As a consequence of this approach is achieved significant acceleration; (2) Except speed of searching and drawing optimal triangulation, this method save the memory in the current buffer (in the operational memory), and in the size of the output file.

Acknowledgements. Predrag Stanimirović gratefully acknowledge the support from Research Project 174013 of the Serbian Ministry of Science.

References

[1] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Speckmann, *On minimum weight pseudo-triangulations*, Computational Geometry-Theory and Applications, 42 (2009), 627-631.

[2] M. Berg, O. Cheong, M. Kreveld, H. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd edition, New York, USA, Springer Verlag, 2008.

[3] B. Chen, H. Cheng, *Interpretive OpenGL for computer graphics*, Computers and Graphics, 29 (2005), 331–339.

[4] J.X. Chen, C. Chen, *Foundations of 3D Graphics Programming: Using JOGL and Java3D*, New York, USA, Springer, 2008.

[5] D. Eppstein, *Approximation of the minimum weight triangulation*, Discrete and Computational Geometry, 11 (1994), 163–191.

[6] M.R. Garey, D.S. Johnson, F.P. Preparata, R.E. Tarjan, *Triangulating a simple polygon*, Information Processing Letters, 7 (1978), 175–180.

[7] D.R. Heffelfinger, *Java EE 6 Development with NetBeans 7,* Birmingham, UK: Pack publishing, 2011.

[8] F. Hurtado, M. Noy, *Graph of triangulations of a convex polygon and tree of triangulations*, Computational Geometry, 13 (1999), 179–188.

[9] Y. Ito, K. Nakano, *A GPU implementation of dynamic programming for the optimal polygon triangulation,* IEICE Transactions on Information and Systems: Special Section on Parallel and Distributed Computing and Networking, 96 (2013), 2596-2603.

[10] J.M. Keil, T.S. Vassilev, *Algorithms for optimal area triangulations of a convex polygon*, Computational Geometry, 35 (2006), 173-187.

[11] F. Klawonn, *Introduction to Computer Graphics: Using Java 2D and 3D. 2nd ed.*, New York, USA, Springer, 2012.

[12] T. Koshy, *Catalan Numbers with Applications*, London, UK: Oxord University Press, 2009.

[13] J.R. Loera, F. Santo, *Triangulations: Structures for Algorithms and Applications*, New York, USA, Springer Verlag, 2003.

[14] T. Mirzoev, S. Vassilev, *New Results on Optimal Area Triangulations of Convex Polygons*, In: Proceedings of the XII Encuentros de Geometria Computacional, Valladolid, Spain, 2007.

[15] W. Mulzer, G. Rote, *Minimum weight triangulation is NP-hard*, Journal ACM, 55 (2008), 1–29.

[16] M. Pistellato, F. Bergamasco, A. Albarelli, A. Torsello, *Dynamic optimal path selection for 3d triangulation with multiple cameras,* In: Conference 18th International Conference on Image Analysis and Processing, Genoa, Italy, 2015, 468-479.

[17] M. Saračević, P. Stanimirović, S. Mašović, E. Biševac, *Implementation of the convex polygon triangulation algorithm,* Facta Universitatis, series: Mathematics and Informatics, 27 (2012), 213–228.

[18] P. Stanimirović, P. Krtolica, M. Saračević, S. Mašović, *Decomposition of Catalan numbers and convex polygon triangulations,* International Journal of Computer Mathematics, 91 (2014), 1315–1328.

[19] A. Yousefi, N. Young, *On a linear program for minimum-weight triangulation,* SIAM journal on computing, 43 (2014), 25–51.

Authors' addresses:

Muzafer Saračević
University of Novi Pazar, Department of Computer Sciences,
Dimitrija Tucovica bb, 36300 Novi Pazar, Serbia.
E-mail: muzafers@uninp.edu.rs

Sead Mašović, Predrag Stanimirović and Predrag Krtolica
University of Niš, Faculty of Sciences and Mathematics,
Višegradska 33, 18000 Niš, Serbia.
E-mail: sead.masovic@gmail.com
pecko@pmf.ni.ac.rs
krca@pmf.ni.ac.rs