FACTA UNIVERSITATIS (NIŠ)
Ser. Math. Inform. Vol. 27 No 2 (2012), 213–228

IMPLEMENTATION OF THE CONVEX POLYGON TRIANGULATION ALGORITHM

Muzafer Saračević, Predrag Stanimirović, Sead Mašović, Enver Biševac

Abstract. Implementations of the algorithm for generating and displaying triangulations of the convex polygon, given by Hurtado and Noy [6], in three programming languages (*Java, Python, C++*) are described and compared. Our main aim is to show the advantages and disadvantages of these programming languages in resolving this useful algorithm in computational geometry and computer graphics. We have performed a comparative analysis of developed programs with respect to three criteria: CPU time spanned for drawing triangulations, the complexity of the source code and the simplicity of the implementation.
Keywords: Triangulation Polygon, Hurtado-Noy hierarchy, Java, Python, C++.

1. Introduction

A triangulation of a simple polygon assumes the decomposition of its interior into triangles, without mutually intersections of internal diagonals. The polygon triangulation is an important aspect of computer graphics and computational geometry. Mainly, triangulation allows a three-dimensional view of objects from the set of points. Triangulation also provides a mechanism for "glazing" 3D, figures which is very important tool for speed, quality and resolution of the objects in computer graphics. In this paper we present implementation of the Hurtado-Noy algorithm for the convex polygon triangulation [6].

We have selected three different programming languages (*Java*, *Python* and *C++*) as different environments for the the Hurtado–Noy method implementation. Some of comparative advantages and disadvantages of these programming languages are discussed. The programming language *Java* can be characterized as: simple, high-performance, object-oriented, multi-threading, dynamic, distributed and secure. The standard technique for *Java* execution is interpretation, which provides for extensive portability of programs. A *Java* interpreter dynamically executes *Java* bytecodes, which comprise the instruction set of the *Java* Virtual Machine (JVM) [15]. The advantage of *Java* language is that most programming languages either

Received May 12, 2012.
2010 *Mathematics Subject Classification.* Primary 68U05; Secondary 68N15, 32B25

interpret or compile to run on a computer, whereas *Java* compiles and interprets simultaneously [12].

Python is also an object-oriented and interpreted programming language and has been created to improve the compiled languages such as *C++* and *Java*. The main advantage of *Python* is its simple syntax and its general simplicity, which allows developers to focus on the problems rather than on the programming. A disadvantage of *Python* is its lack of speed in the case when the starting activity requires considerable processing power. The first observation in the comparison of *Python* and *Java* is usually the easier work in *Python*. Compared to *Java*, *Python* applications that implement the same algorithm are much shorter. *Python*'s productivity is at the very high level, primarily because of the syntax and especially because of the large number of ready-made libraries and modules that are available in the standard distribution [3]. Runtime implementation of *Python* language is available on many platforms. Unlike *Java*, *Python*'s syntax does not require encapsulation of the class and does not support real interfaces [5]. However, it supports multiple inheritance and functional programming. *Java* and *C++* are very similar in syntax.

C++ is more consistent as a language and during the learning process. It is the closest to the minimal and uncontradicted language. *Java* is a static type–checked language which offers performance, robustness and modularity as such, while *Python* is a run-time type-checked language which offers rapid prototyping, dynamic run-time modification and delayed evaluation.

This paper is organized as following. In the next section we enumerate several related applications in computer graphics and computational geometry, written in the mentioned programming languages. In the third section we describe the Hurtado-Noy algorithm for a convex polygon triangulation. The fourth section includes implementations of this algorithm in *Java*, *Python* and *C++*. We have described some of methods and classes defined in developed programs. The fifth part of our paper includes a comparative analysis of obtained numerical results.

2. Related research

In this section we list some applications of the programming languages *Java*, *Python* and *C++* in computer graphics.

The authors of the paper [1] considered the design of several *Java* applets that visualize how the Voronoi diagram for the n-gon triangulation continuously changes as individual points are moved across the plane, or as the underlying distance function is changed. Moreover, the authors report some experiences of using these *Java* applets in teaching and research. The paper [2] specifies that the compound 3D visualization modeling system can be built up with the open-sourced graphic libraries PYOpenGL and VTK (Visualization Toolkit). Spatial discrete points of the Delaunay triangulation in any plane are accomplished through the setting of the projection plane. Taking full advantage of the topology characteristic, a kind of algorithm which can search interrelated triangles, segments and vertexes efficiently is obtained and the object of inserting any constraint is reached.

The paper [4] describes an application of the *Python* language in computational geometry, built on top of the *Java* language and run-time environment. Implementing *Python* in *Java* has a number of limitations when compared to the current implementation of *Python*. Advantages of programming language *C++* in computational geometry and computer graphics are described in [3]. The authors of paper [5] present how to take the best from the programming languages *Java* and *Python* in software development and utilization of *Java Python* Interface.

We obtained the motivation for our comparative analysis for triangulation algorithm of the polygon [6] by exploring a couple of similar tests for some related algorithms. Testing results of several programming languages for more algorithms are stated on the portal - *The Computer Language Benchmarks*. In order to compare testing results with our implementation of the algorithm in above mentioned programming languages we took the algorithm based on BinaryTrees. The reason why we have chosen this structure is for the simple reason, because can find its application in the process of triangulation of convex polygons. Figure 2.1 shows performance measurements for *Java*, *Python* and *C++* program for BinaryTrees.

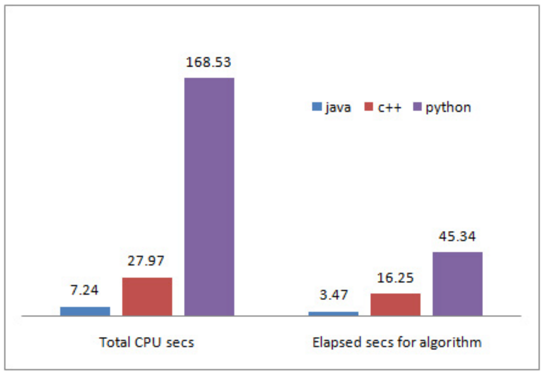

FIG. 2.1: Performance measurements for algorithm - *BinaryTrees*

As for the CPU Load (testing on this portal is carried out on Intel Q6600 quad-core) and in the most demanding case where is $n = 20$, highest CPU occupation is registered by *Python* (93% , 92% , 98%, 93%). The values in parentheses refer to the core CPU. In the case for *C++* (27% ,98%, 25%, 24%) and *Java* (55%, 27% , 76% , 57%).

3. Hurtado-Noy algorithm for the convex polygon triangulation

The number of convex polygon triangulations is closely related to Catalan number. The nth Catalan number C_n is defined by

$$C_n = \frac{(2n)!}{(n+1)!n!}. \quad (3.1)$$

The Hurtado-Noy algorithm (Algorithm 3..1) for triangulation of a convex polygon is described in [6].

Algorithm 3..1 Hurtado-Noy algorithm

```
Require: Positive integer $n$
1: Check the structure containing $2n-5$ vertex pairs looking for pairs $(i_k, n-1)$, $i_k \in \{1, 2, \ldots, n-2\}$, $2 \leqslant k \leqslant n-2$, i.e. diagonals incident to vertex $n-1$. The positions of these indices $i_k$ within the structure describing a triangulation should be stored in the array.
2: For every $i_k$ perform the transformation $(i_l, n-1) \to (i_l, n)$, $i_l < i_k$, $0 \leqslant l \leqslant n-3$.
3: Insert new pairs $(i_k, n)$ and $(n-1, n)$ into the structure.
4: Take next $i_k$, if any, and go to Step (2).
5: Continue the above procedure with next $(n-1)$-gon triangulation (i.e. structure with $2n-5$ vertex pairs) if any. Otherwise halt.
```

Let $T(n)$ be the set of triangulations of an n-gon. Every triangulation t that belong to $T(n)$ has a "father" in $T(n-1)$ and one or more "sons" in the set of triangulations $T(n+1)$. For given set $T(n)$, we can actually generate triangulations of the $(n+1)$-gon arising from arbitrary triangulation $t \in T(n)$. The number of sons of t is dependent on the out degree of the vertex V_n [7]. What we are basically doing is opening the parent polygon and adding in the $(n+1)$th vertex. We keep all the edges of the parent that did not involve V_n as they were. Next we remove all the edges of the form $E(p, n)$ from the parent and add in $E(p, n+1)$ instead. Thereafter we add in all edges $E(p, n)$ of the parent, where p is a vertex numbered higher than i (remember we are constructing the ith son). This principle is illustrated on Figure 3.1.

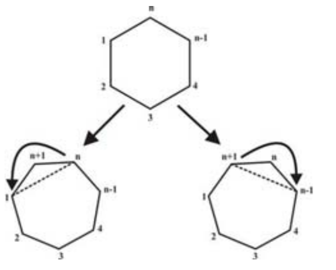

FIG. 3.1: The way of forming the new triangulation for $(n+1)$-gon, according to Hurtado algorithm

Indeed, we even need the triangulations of the $(n-1)$th polygon ready. And then we generate d sons for each of those, where d is the degree of V_n of each triangulation.

However the hierarchy is nevertheless important because of its inherent simplicity and also owing to the fact that it has a number of really exciting properties which we shall state below (Figure 3.2).

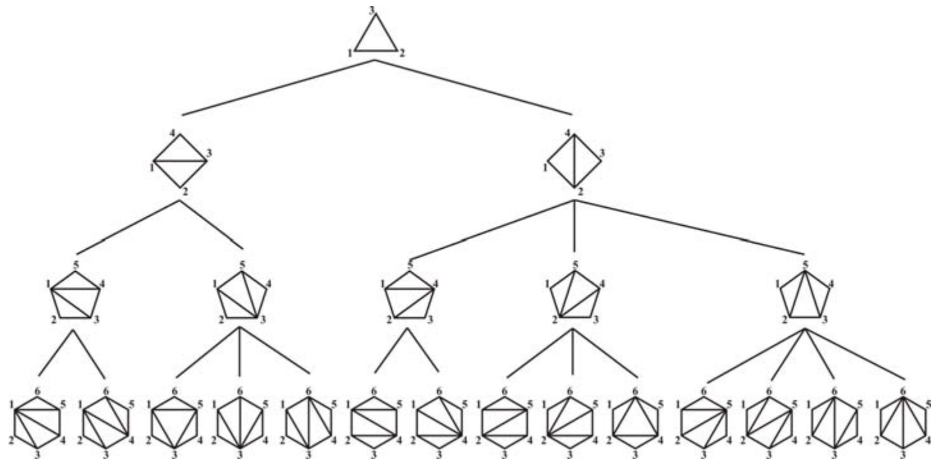

FIG. 3.2: Levels three to six of the tree of triangulations - Hurtado Noy Hierarchy

4. Implementations in *Java*, *Python* and *C++*

First, we list some applications of these programming languages in terms of working with graphic elements and solving algorithms in computational geometry.

The *Java* computational geometry library contains an implementation of major computational geometry algorithms in *Java* [10], in a similar way that the Computational Geometry Algorithms Library (CGAL) does for the *C++* language. In recent years, *Java* has significantly improved, particularly since the appearance of *Java* 2 standard. *Java* programming language possesses many opportunities to create applications with interactive graphical user interface (GUI), a detailed image processing and programming of graphical elements.

The *Computational Geometry Algorithms Library* (CGAL) in *C++* is used in various areas requiring geometric computation, such as computer graphics, scientific visualization, computer aided design and modeling, geographic information systems, molecular biology, medical imaging, robotics and motion planning, mesh generation and numerical methods. Real/Expr is a set of *C++* class libraries which supports the precision-driven approach in the implementation of certain algorithms in computational geometry.

The implementation of the Hurtado-Noy algorithm is realized through the classes: `GenerateTriangulations, Triangulation, Node, LeafNode, Point` and `PostScriptWriter` (Figure 4.1).

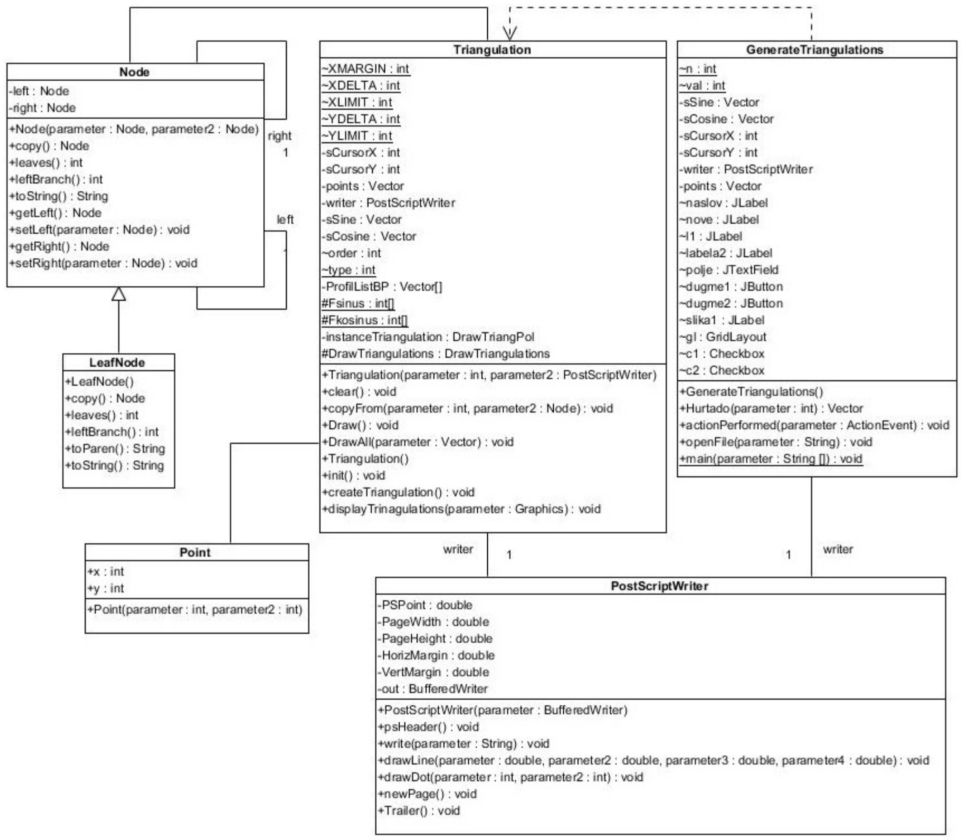

Fig. 4.1: UML Class Diagram

In this part of the paper we present a comparative view for implementation of the algorithm through selected segments of source code. We describe main parts of the class `Triangulation`, including the methods `Draw` and `DrawAll` and executive class `GenerateTriangulations` and `Hurtado` method. Due to the ease of comparison of programming languages, source codes are divided into 5 parts (parts A, B, C, D, E in Appendix). The class `Triangulation` is responsible for displaying a convex polygon triangulation. This class provides the verification of all the vertices of the polygon. The source code in all programming languages for the class `Triangulation` is presented in Appendix A.

Method `Draw` is located in the class `Triangulation` and presented in Appendix B. This method is responsible for making an individual triangulation. The command `drawLine` is used to obtain the appropriate number of vertices to form the regular convex polygon (see Appendix B1 and B2). One combination of internal diagonals forms a triangulation of the convex polygon. Vectors `sSine` and `sCosine` and variable d provide drawing regular convex polygons. The implementation takes respectively those triangulations of internal diagonal which form the triangles inside the polygon provided that they do not intersect (Algorithm 4..1).

Algorithm 4..1 Finding all internal diagonals of a simple convex polygons

```
1: Step 1: Set the counter $i = 1$.
2: Step 2: Connect $i$th point with $(i + 2)$th point,
3: Step 3: Check wheter the new diagonal is internal.
       Yes: Add it in the list and eliminate $(i + 1)$th point of the polygon.
       No: $i = i + 1$.
4: Step 4: Return to Step 2.
```

Method `DrawAll` (Appendix C) also belongs to the class `Triangulation` and it provides iteration where the method `Draw` is called as many times as it is necessary for a given value n in the class `GenerateTriangulations`. And for drawing regular convex polygons it calls the class Node (Appendix C1, C2 and C3). The method `copyFrom(int aOffset, Node t)` provides insertion of new vertices in the left directions `copyFrom(aOffset, t.getLeft())`. The insertion is done by the command `points.add(new Point( aOffset, aOffset+t.leaves()))`.

The `Hurtado` method, described in Appendix D, creates string of objects of the class Node. These are vertices of polygons used to create instances of the class `Triangulation` with the argument n (number of vertices of the polygon for which is carried out by displaying the triangulation).

In *Java*, *C++* and *Python*, the order of triangulation by the Hurtado-Noy hierarchy (Appendix D1 and D2) is arranged. The main executive method (Appendix E) in *Java* is located in the class `GenerateTriangulations` and is responsible for the basic input of n parameter (number of vertices of a convex polygon). This method for *Java*, *Python* and *C++* stores all drawn triangulation in output file using the class `PostSriptWriter` (Appendix E1 and E2). The method `writer.psHeader()` ensures that all triangulations are recorded in graphic representation in PS format. The main executive method of the class `Triangulation` uses the `DrawAll()` method which draws all triangulations by the Hurtado order `(mPicture.DrawAll(btrees))`.

Instance of `Triangulation` class `mPicture` and `btrees` are object type `Vector`, which with command `app.Hurtado(n-2)` perform `GenerateTriangulations` class. In *Python* the special method of `setOptionParser()` is implemented, which implies the main parameter of n and it also defines the mode of execution of the main executive method (Appendix E2). As an example of the application execution, we present the output in Java for $n = 3, 4, 5, 6$ (see Figure 4.2).

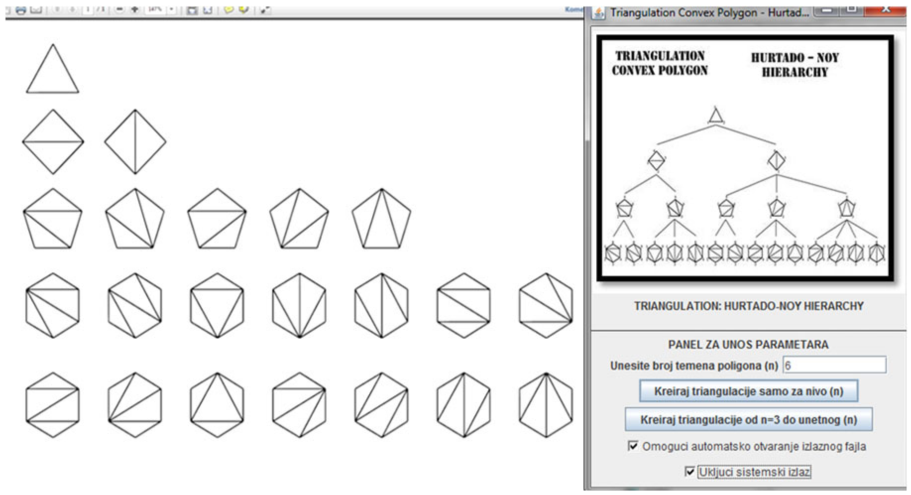
Triangulation Convex Polygon - Hurtad...
TRIANGULATION CONVEX POLYGON
HURTADO - NOY HIERARCHY
TRIANGULATION: HURTADO-NOY HIERARCHY
PANEL ZA UNOS PARAMETARA
Unesite broj temena poligona (n) 6
Kreiraj triangulacije samo za nivo (n)
Kreiraj triangulacije od n=3 do unetnog (n)
Omoguci automatsko otvaranje izlaznog fajla
Ukljuci sistemski izlaz

FIG. 4.2: Graphic display of the Hurtado-Noy order in the case $n = 3, 4, 5, 6$

5. Comparative analysis

In this part of the paper we compare implementations of the Hurtado's algorithm in three different programming languages. The speed of execution and simplicity of executable source code are used in the comparative analysis. Table 5.1 contains a comparative analysis for all performed tests. Taken values for n are integers from 10 to 16. We compared the speed of plotting the triangulations as well as the number of generated triangulations per second.

Table 5.1: Comparative analysis for all performed tests

Criterion	n	***Java***	***Python***	***C++***
Speed of plotting the triangulations	10	2.96	4.19	2.95
	11	4.07	6.08	4.25
	12	5.81	17.83	6.23
	13	15.24	66.25	15.88
	14	46.34	244.52	55.28
	15	124.18	886.12	134.12
	16	328.16	3185.13	399.54
The number of combinations per second	10	483.11	341.29	484.75
	11	1194.59	799.67	1144.00
	12	2890.88	942.01	2695.99
	13	3857.35	887.34	3701.89
	14	4488.82	850.70	3762.88
	15	5982.44	838.37	5539.07
	16	8149.80	839.66	6693.80

*PC performance for testing: CPU: Intel(R) Core2Duo, T7700(2.40GHz), L2 Cache 4MB (Full-Speed), RAM Memory - 2Gb, Graphic card: NVIDIA GeForce 8600 MGS.

It can be observed that increasing values of n the *Java* application increases the number of generated triangulations per second. On the other hand, in the *Python* application the number of generated triangulations per second decreases for values $n > 12$. Similarly to *Java*, the *C++* code increases the number of plotted triangulations in seconds with increasing the number of vertices up to $n = 13$. However, for subsequent values of n we observed the stagnation in *Python* and partially in *C++*, which is not the case in *Java*. *C++* appears in this test as a bit faster than *Java* but much faster than *Python* (Figure 5.1).

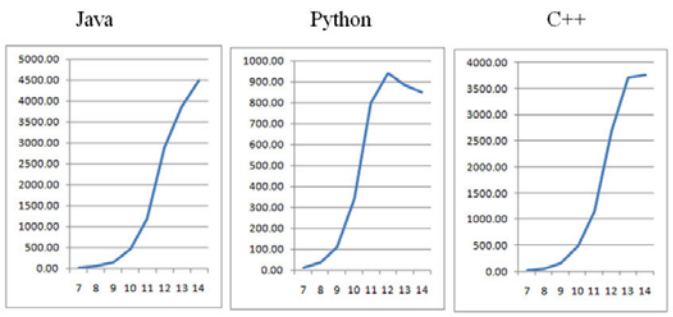

FIG. 5.1: The number of generated triangulations per second

Comparing the speed of execution, we come to the conclusion that *Java* needs the shortest time necessary to draw triangulations for a given value n. The next shortest time shows the implementation in *C++*. Obtained results are illustrated on Figure 5.2. The x axis contains values for the number n, while the y axis contains values for the spanned CPU time (in seconds).

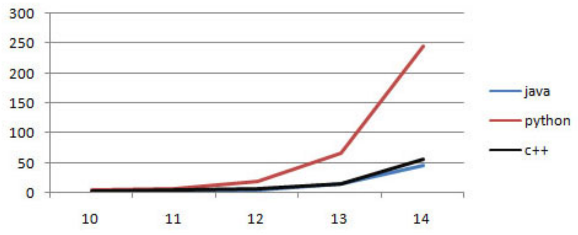

FIG. 5.2: The ratio speed of execution

When we take a cross section of all graphics for the number of triangulations per second for the values $n = 10, 11, 12, 13, 14$, it can be noted that in *Java* and *C++* line realize a number of drawn triangulations per second continuously in an ascending path. In *Python*, up to value $n = 12$, the graph line is in an ascending path and after that it is decreasing. Speed ratio is presented in Figure 5.3. The vertical axis of the graphical representation contains the number of displayed triangulations per second while the horizontal axis contains values for n.

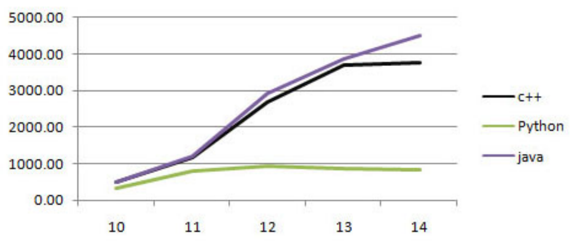

Fig. 5.3: The number of triangulations per second individually

The *Java* application *Java Virtual Machine* (JVM) reserves more working memory during the execution for its object, while *Python* at the time of forming an instance of the class `Triangulation` takes much less memory. Compiled *C++* code is directly translated into machine language but because of the connection between classes separated into files requiring much more space than byte code. JVM is compiled for many platforms and it comes in a package with the standard *Java* libraries that together make *Java Runtime Environment* (JRE). Any platform where it can be installed can run *Java* applications. This is a huge advantage, it shortens the time of porting to other platforms whilst giving extra control in the execution of code and provides greater security.
All three applications can be downloaded from:
http://muzafers.uninp.edu.rs/triangulacija.html

6. Conclusion

The presented results are used to compare three programming language (*Java*, *C++* or *Python*) in solving an important algorithm in computational geometry. We give a proposal which programming language is most suitable to implement algorithms for a convex polygon triangulation by Hurtado-Noy algorithm in terms of speed, simplicity of syntax and clarity of the source code.

Our experience is that *Java* can be identified as a programming language with exceptional abilities when it comes to working with graphics and also when it comes to speed of execution. Object modeling and encapsulation of real object is the best in *C++*. This language has gone furthest in the support of the object-oriented programming. *Python* is a programming language characterized by a simple syntax and clarity but a relatively slower execution. On the other hand, *Java* also has object-oriented programming language with a simpler syntax and better portability than the other two languages.

REFERENCES

1. C. Icking, R. Klein, P. Kllner, L. Ma: *Java Applets for the Dynamic Visualization of Voronoi Diagrams*, Lecture Notes In Computer Science **2598** (2003), 191–205.
2. L. Zhen-ping, H. Huai-jian, L. Qiang, Z. Fa-hua: *Study of the technology of 3D modeling and visualization system based on Python*, Changjiang Water Resources Commission, Wuhan 430070, China
3. T. Suzumura, S. Trent, M. Tatsubori, A. Tozawa, T. Onodera *Performance Comparison of Web Service Engines in PHP, Java and C, Web Services*, 2008. ICWS '08. IEEE International Conference on , 23-26 Sept. 2008, 385–392
4. J. Hugunin,: *Python and Java: The Best of BothWorlds*, Corporation for National Research Initiatives
5. D. Cunningham, E. Subrahmanian, A. Westerberg: *User-Centered Evolutionary Software Development Using Python and Java*, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA
6. F. Hurtado, M. Noy: *Graph of Triangulations of a Convex Polygon and tree of triangulations*, Comput. Geom. **13** (1999), 179–188.
7. F. Hurtado, M. Noy: *Counting triangulations of almost-convex polygons*, Ars Combinatoria **45** (1997) 169- 179.
8. F. Hurtado, M. Noy: *Ears of triangulations and Catalan numbers*, Discrete Math. **149** (1996), 319-324.
9. M.R. Garey, D.S. Johnson, F.P. Preparata, R.E. Tarjan: *Triangulating a simple polygon*, Inform. Process. Lett. **7** (1978), 175-180.
10. I. Peter and S. Gumhold: *Teaching computer graphics with java 3*, WSI/GRIS, University of Tbingen, Germany.
11. J. Mark Keil, S.T. Vassilev: *An Algorithm for the MaxMin Area Triangulation of a Convex Polygon*, 15th Canadian Conference on Computational Geometry, CCCG 2003, Halifax, Nova Scotia, August 11
12. S. Masovic, M. Saracevic, H. Kamberovic, M. Kudumovic: *Java technology in the design and implementation of web applications*, TTEM - Technics Technologies Education Management, **7**, No.2, (2012).
13. M.J. Laszlo: *Computational geometry and computer graphics in C++*, Prentice Hall (Upper Saddle River, N.J.), Book (ISBN 0132908425), 1996
14. M. Saracevic, P. Stanimirovic, S. Masovic: *Implementation of some algorithms in computer graphics in Java*, TTEM-Technics Technologies Education Management,(2012), Accepted.
15. I. Kazi, H. Chen, B. Stanley, D. Lilja: *Techniques for Obtaining High Performance in Java Programs*, ACM Computing Surveys, **32**, No.3, pp.213-240, (2000).

APPENDIX

A) Class Triangulation

A1 - Java

```
public class Triangulation {
public Triangulation(int edges, PostScriptWriter writer) {
           this.sCursorX = Triangulation.XLIMIT;
           this.sCursorY = Triangulation.YLIMIT;
           this.points = new Vector<Point>();
           this.writer = writer;
           this.sSine = new Vector();
           this.sCosine = new Vector();
for (int k=0; k < edges; k++) {
              double d = (4*k+edges)*Math.PI/(2*edges);
              this.sSine.add(30*Math.sin(d));
              this.sCosine.add(30*Math.cos(d)); }
        }
```

A2 - Python

```
class Triangulation(object):
  def __init__(self,edges,writer):
    self.sCursorX = Triangulation.XLIMIT
    self.sCursorY = Triangulation.YLIMIT
    self.points = []
    self.writer = writer
    self.sSine = [] self.sCosine = []
    for k in range(edges):
      d = (4*k+edges)*math.pi/(2*edges)
      self.sSine.append(30*math.sin(d))
      self.sCosine.append(30*math.cos(d)
```

A3 - C++

```
Triangulation::Triangulation(int edges, PostScriptWriter *writer){
        this->sCursorX = Triangulation::XLIMIT;
        this->sCursorY = Triangulation::YLIMIT;
        this->points = new Vector<Point*>();
        this->writer = writer;
        this->sSine = new Vector();
        this->sCosine = new Vector();
        for (int k = 0; k < edges; k++) {
                  double d = (4*k + edges)*M_PI / (2*edges);
                  this->sSine->add(30*sin(d));
                  this->sCosine->add(30*cos(d)); }
}
```

B)Method Draw

B1 - Java

```
public void Draw() throws IOException {
if (Triangulation.XLIMIT <= this.sCursorX) {
        this.sCursorX = Triangulation.XMARGIN;
        this.sCursorY -= Triangulation.YDELTA; }
    for (int i = 0; i < this.points.size(); i++) {
        Point p = this.points.get(i);
        this.writer.drawLine (
        -this.sCosine.get(p.x)+this.sCursorX,
         this.sSine.get(p.x)+this.sCursorY,
        -this.sCosine.get(p.y)+this.sCursorX,
         this.sSine.get(p.y)+this.sCursorY); }
}
```

B2 - Python

```
def Draw(self):
    if ( Triangulation.XLIMIT <= self.sCursorX ):
    self.sCursorX = Triangulation.XMARGIN
    self.sCursorY -= Triangulation.YDELTA
        for p in self.points:
        self.writer.drawLine (
        -self.sCosine[p.x]+self.sCursorX,
         self.sSine[p.x]+self.sCursorY,
        -self.sCosine[p.y]+self.sCursorX,
         self.sSine[p.y]+self.sCursorY )
        self.sCursorX += Triangulation.XDELTA
```

C) Method DrawAll

C1 - Java

```
public void DrawAll(Vector<Node> trees) throws IOException {
this.writer.psHeader();
for (int i = 0; i < trees.size(); i++) {
Node t = trees.get(i);
        this.clear();
        this.copyFrom(0, t);
        this.Draw();
    if (i != 0 && i % 55 == 0) {
       this.sCursorX = Triangulation.XLIMIT;
       this.sCursorY = Triangulation.YLIMIT;
       this.writer.newPage(); }
                }
}
```

C2 - Python

```
def DrawAll(self,trees):
    self.writer.psHeader()
    count = 0
    for t in trees:
      self.clear()
      self.copyFrom( 0, t );
      count += 1
      self.Draw();
      if count != 0 and count % 55 == 0:
        self.sCursorX = Triangulation.XLIMIT
        self.sCursorY = Triangulation.YLIMIT
        self.writer.new_page()
    self.writer.Trailer()
```

C3 - C++

```
    void:DrawAll(Vector<Node*> *trees) throw(IOException){
        this->writer->psHeader();
        for (int i = 0; i < trees->size(); i++){
            Node *t = trees->get(i);
                this->clear();
            this->copyFrom(0, t);
            this->Draw();
            if (i != 0 && i % 55 == 0){
                this->sCursorX = Triangulation::XLIMIT;
                this->sCursorY = Triangulation::YLIMIT;
                this->writer->newPage(); }
            }
}
```

D) Method Hurtado

D1 - Java

```
public Vector<Node> Hurtado(int limit) {
Vector<Vector> trees = new Vector<Vector>();
    trees.add(new Vector<LeafNode>());
    trees.get(0).add(new LeafNode());
        for (int n=0; n < limit; n++) {
            Vector<Node> level = new Vector();
        for (int q = 0; q < trees.get(n).size(); q++) {
            Node t = (Node)trees.get(n).get(q);
            Node s = new Node(new LeafNode(), t.copy());
            level.add(s);
        for (int k = 0; k < t.leftBranch(); k++) {
            s = t.copy();
            Node r = s;
        for (int i=0; i < k; i++)
            s = s.getLeft();
            s.setLeft(new Node(new LeafNode(), s.getLeft()));
            level.add(r); }
                }
    trees.add(level); }
    return trees.get(limit);
}
```

D2 - Python

```
def Hurtado(limit):
    trees = [[LeafNode()]]
    for n in range(limit):
        level = []
    for t in trees[n]:
        s = Node(LeafNode(),t.copy())
        level.append(s)
    for k in range(t.leftbranch()):
        s = t.copy() r = s
    for i in range(k):
        s = s.left
        s.left = Node(LeafNode(),s.left)
        level.append(r);
    trees.append(level)
    return trees[limit]
```

E) Main method

E1 - Java

```
public static void main(String args[]){
    try{
        int n; // number of vertices
        FileWriter fstream = new FileWriter(n+"-polygons.ps");
        BufferedWriter out = new BufferedWriter(fstream);
        PostScriptWriter writer = new PostScriptWriter(out);
        App app = new App();
        Vector<Node> btrees = app.Hurtado(n-2);
        Triangulation mPicture = new Triangulation(n,writer);
            mPicture.DrawAll(btrees);
            out.close();
        }
    catch (Exception e){
    e.printStackTrace(); }
        }
}
```

E2 - Python

```
def main():
    global DEBUG
    parser = setOptionParser()
    (options,_) = parser.parse_args()
    DEBUG = options.debug
    writer = PostScriptWriter(fh)
    btrees = Hurtado(options.edges-2)
    if DEBUG:
        print len(btrees)
    for t in btrees:
        print t.toParen(),str(t)
        print len(btrees)
    mPicture = Triangulation(options.edges,writer)
    mPicture.DrawAll(btrees)
    fh.close()

def setOptionParser():
    parser = OptionParser(usage=usage)
    parser.add_option('-n','--edges',type="int",default=3)
    return parser
```

Muzafer H. Saračević
Faculty of Sciences and Mathematics
Department of Mathematics and Informatics
Višegradska 33
18000 Niš, Serbia
muzafers@gmail.com

Predrag S. Stanimirović
Faculty of Sciences and Mathematics
Department of Mathematics and Informatics
Višegradska 33
18000 Niš, Serbia
pecko@pmf.ni.ac.rs

Sead H. Mašović
Faculty of Sciences and Mathematics
Department of Mathematics and Informatics
Višegradska 33
18000 Niš, Serbia
sead.masovic@gmail.com

Enver Biševac
University of Novi Pazar
Department of Computer Science
Dimitrija Tucovića bb
36300 Novi Pazar
e.bisevac@uninp.edu.rs