COMPUTER-AIDED DESIGN & APPLICATIONS, 2016
VOL. 13, NO. 3, 389–396
http://dx.doi.org/10.1080/16864360.2015.1114397

IFOG: Inductive Functional Programming for Geometric Processing

Masaji Tanaka[1], Yuki Takamiya[1], Naoki Tsubota[1] and Kenzo Iwama[2]

[1]Okayama University of Science, Japan; [2]EngiCom Corporation, Japan

ABSTRACT

Since decades, especially in CAD and CG, to solve various kinds of problems and/or to develop automatic systems, not only geometric modeling techniques but also combinatorial searches of geometric elements such as line segments have been applied extensively. Generally it is troublesome and time consuming to program the combinatorial searches for programmers because they are basically algorithmic and it would be difficult to formalize them. In this paper, a new programming technique called IFOG (Inductive Functional prOgramming for Geometric processing) is proposed. IFOG enables to realize easier programing for programmers, especially for beginners, than conventional programming techniques for geometric processing. In IFOG, geometric elements are expressed as their properties, and they are also instances of geometric classes that can be generalized from the instances inductively. Since the classes and instances are stored as text files in a PC, programmers can read and write them whenever they develop programs in IFOG. Therefore, they do not have to grasp the whole data of the relationships of geometric elements temporarily in their brains in their programming. The effectiveness of IFOG is indicated by using practical examples in this paper, and it has been verified by our experimental system.

KEYWORDS

IFOG; inductive programming; combinatorial search; geometric processing

1. Introduction

Since decades, especially in CAD and CG, to solve various kinds of problems and/or to develop automatic systems, not only geometric modeling techniques but also combinatorial searches of geometric elements such as line segments have been applied extensively. For example, to develop automatic systems that can convert 2D drawings into 3D models, a great many recognitions of complex geometric elements such as primitives and features are required for their programming, and each of the recognitions would be programed as combinatorial searches of simple geometric elements such as lines and faces, e.g. [7,8]. Generally it is troublesome and time consuming to program the combinatorial searches for programmers because of the following reason. First, they are basically algorithmic. Second, their formalization would be difficult because there are few mathematical bases in them although NP-hard or NP-complete problems exist in computational complexity theory. If a programmer wants to detect each parallelogram from a 2D drawing drawn in a 2D CAD system, firstly he/she might search four straight lines, and then calculate their relationships. As the result, his/her programs for the detection would consist of too many procedures by using conventional programming techniques.

On the other hand, in object-oriented programming (OOP), various kinds of classes can be defined, and programs almost consist of passing messages among objects as instances of the classes, e.g. [1]. So the class of straight lines can be made in OOP. However, this class must be defined by programmers before making a program to search parallelograms from 2D drawings. Continuously if the other geometric elements such as Y-junctions of lines are required to search in the drawings, the class of straight lines would be modified. This modification would influence the program comprehensively. Therefore, in this case, the definition of the class of straight lines would be difficult for beginners.

In this paper, a new programming technique called IFOG (Inductive Functional prOgramming for Geometric processing) is proposed. IFOG enables to realize easier programing for programmers, especially for beginners, than conventional programming techniques for geometric processing. The basic idea of IFOG is as follows. Suppose that a little child has already known how to draw a straight line in a paper by using a pen. A teacher

CONTACT Masaji Tanaka tanaka@mech.ous.ac.jp

could teach him/her which line is longer than another line inductively by indicating plural examples of two straight lines drawn in papers. Generally the length of a straight line and the comparison of length of two lines are knowledge for humans and also the properties of a line. Continuously, when he/she learns about advanced geometry such as triangles, the relationships among straight lines and the properties of them would be increased and understood as knowledge step by step. This learning process in a human is applied in IFOG. Suppose a programming beginner makes a program that can draw simple line drawings by users. Firstly he/she would make a program that can draw a straight line in the monitor of a PC. Then he/she would extend the program to be able to trim lines, draw circles and arcs, etc. In the process of making the program, many kinds of data might be stored in many arrays. However, since these arrays would not be summarized, it would be difficult to apply the program by the other programmers. Moreover, when the program becomes too large, he/she would cost much effort to handle too many arrays. On the other hand, if he/she uses OOP for the programming, it would be difficult to make classes of geometric elements because the definitions of them might be modified whenever his/her program becomes complex.

In IFOG, each class can be generalized from its instances inductively. Each instance consists of properties. Also, each class can make its instances. In IFOG, when a programming beginner programs a program that can draw simple line drawing by users, firstly he/she can make the class of points and the class of straight lines. Suppose the contents of the class of points are point number and x-y coordinates. Continuously when his/her program can operate trimming of lines, the property expressing whether a point is an intersection can be added to the class of points in IFOG. Each class is stored as a text file, and its instances are also stored as a text file in a PC in IFOG. Since he/she can read and write them whenever he/she requires some data, he/she does not have to grasp the whole data of the relationships of geometric elements temporarily in his/her brain in IFOG. Moreover, it would be easy to apply the program to the other programs by the other programmers because their first task is mainly reading the text files. We have verified the effectiveness of IFOG by our experimental system that is implemented by Visual C++. The rest of this paper is organized as follows. In Section 2, related works of this paper are described. In Section 3, main algorithm of IFOG is explained. In Section 4, an example program of IFOG is indicated. In Section 5, we discuss the ability of IFOG. In Section 7, we make our conclusions.

2. Related Works

The original idea of IFOG was inspired from one of the authors' machine learning methods, e.g. [2,4]. In the methods, the processes of solving mathematical problems in children were generalized inductively. Also, we developed methods for automatically restoring omitted 2D mechanical drawings by applying the methods, e.g. [5,6]. In these two methods, each geometric element was defined as various kinds of properties. For example, the properties of a straight line can be its position, length, slope, etc., and the properties of the relationship of two straight line segments can be intersection, parallelism, etc. When plural examples of the restorations in 2D drawings are generalized, several properties of lines can be changed into variables. After this generalization, if a new question that is an omitted 2D drawing is input to the methods, its solution would be automatically output. Therefore, when various kinds of problems are applied to the methods, the amount of properties that must be already set would be exploded rapidly. It was a serious issue in our past methods. IFOG solves this issue by controlling the increase of the properties step by step when programs become complex by programmers.

The other related works are as follows. The difference between IFOG and OOP is already described in Section 1. Since instances are expressed in text format in XML (Extensible Markup Language), e.g. [3], it might be similar to IFOG. However, each instance has to be described initially by users in XML. In IFOG, a class can be generalized from its instances and also makes its instances while programs are executed. On the other hand, there are many researches that relate with the expression ways of various kinds of geometric modeling, e.g. [9]. Although they are useful in present CAD systems, basically properties in IFOG are different from them because the properties express characters of geometric elements as in humans.

3. Main Algorithm of IFOG

At the present step, we handle only 2D straight lines as geometric elements. Straight lines are called lines in this paper. They are drawn in a 2D CAD system and each drawing is filed as a DXF(Drawing Exchange Format) file. When a programming beginner programs a program in IFOG, the algorithm of his/her programming can be modeled and summarized as follows.

(1) Programming of a function that makes a text file expressing initial class list.
(2) Programming of a function that makes initial classes and their instances.

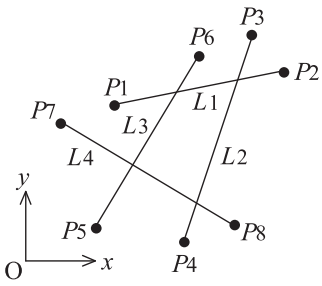


Figure 1. Example 1.

(3) Programming of function(s) that operate the instances.
(4) Programming of a function that can update or define classes.
(5) Programming of a function that can update instances if their classes are updated.
(6) Programming of a function that can update the class list.

In IFOG, a program consists of functions and each geometric element is expressed as properties. Some of the functions and properties can be knowledge for humans. So, he/she can handle the knowledge easily for developing advanced applications in IFOG. This algorithm is practically explained in detail as follows. Fig. 1 illustrates Example 1 that is a 2D line drawing. In this figure, four lines ($L1$, $L2$, $L3$, $L4$) are drawn and their end points ($P1$, $P2$, ... $P8$) are emphasized. The information of these points and lines such as x-y coordinates can be extracted easily from the DXF file of Example 1.

In Step (1), a function that makes ClassList.txt is programed as in Fig. 2. In this file, two classes and their last numbers of instances are described. In Step (2), a function that makes Class_Point.txt, Class_Line.txt, Point.txt and Line.txt is programed as in Fig. 2. The point number and x-y coordinates of each point are properties in Class_Point.txt. In this file, the variable type and the name of each property is indicated such as "No.: int t1". Especially double numbers are rounded off moderately in the figures of this paper because it is meaningless. Also, the line number and two numbers of end points of each line are properties in Class_line.txt. Two instance files

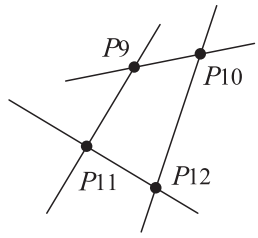


Figure 3. Intersections in Example 1.

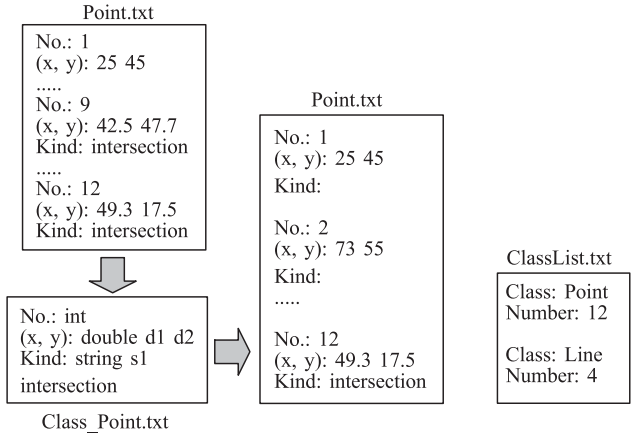


Figure 4. Updating files in Example 1.

(Point.txt and Line.txt) are made from the two classes respectively as in Fig. 2. In these files, middle parts are omitted.

If a programming beginner wants to detect all intersections in Example 1, he/she can program a function that obtains all intersections of four lines in Step (3). Fig. 3 illustrates them ($P9$, $P10$, $P11$, $P12$). By the function, since these intersections are points, they are stored in Point.txt as in the top left side of Fig. 4. In this file, a property "Kind" is added to $P9$ and $P12$. These additions can be programed in the function of Step (3). In Step (4), a function that can add some property to existing classes or define new classes is programed. So, "Kind" is added to Point class as in the bottom left side of Fig. 4. The variable type of "Kind" can be generalized from the real values of "Kind" in four instances ($P9$, $P10$, $P11$, $P12$). In this case, since all real values are "intersection", the

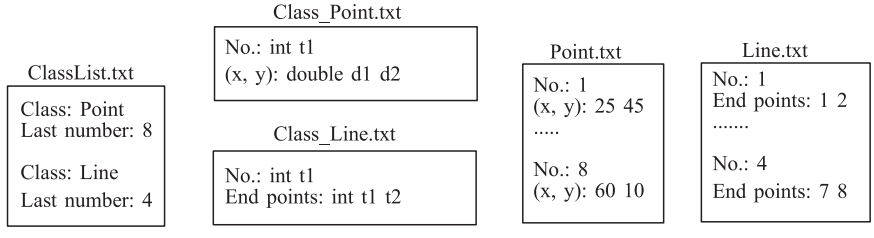


Figure 2. Initial files of Example 1.

variable type of "Kind" becomes "string" and its name is "s1". Also, "intersection" is indicated as a real value under "Kind: string s1". In Step (5), a function that can add new properties to all of existing instances in accordance with updated classes is programed. So, "Kind" is added to all instances of Point class as in the center of Fig. 4. In Step (6), a function that can update ClassList.txt is programed as in the right side of Fig. 4. Here, let the name of this program be Recognition_of_intersections.cpp. The program can be expressed as follows.

```
// Recogniton_of_intersections.cpp
main()
{ MakeClassList();//
  Programed in Step (1)
  MakeInitialClassInstance();//
  Programed in Step (2)
  Recognize_Intersection();//
  Programed in Step (3)
  UpdateClass();//
  Programed in Step (4)
  UpdateInstance();//
  Programed in Step (5)
  UpdateClassList();//
  Programed in Step (6)
}
```

The program could express the knowledge of finding intersections of lines in humans. When the program is executed to the other 2D drawings, they can obtain the information about intersections of lines automatically. Therefore, they would become more intelligent than initial states of them.

4. Example of IFOG

In IFOG, CAD data could become more intelligent by executing various kinds of programs. Therefore, high level applications could be developed in IFOG by using intelligent CAD data by programmers, especially programming beginners. In this section, an example of the applications is explained that can divide a 2D line drawing into regions each of which is a closed loop of lines. In general, to program the division would be very difficult for programming beginners because a lot of arrays and very complex loops are required in their programming. In IFOG, smarter programming for the division than conventional programmings would be realized as follows. Fig. 5 illustrates Example 2. In this figure, all numbers of points ($P1$, $P2$, ... $P7$) and lines ($L1$, $L2$, ... , $L7$) are indicated. The main division processes of Example 2 into regions are as follows.

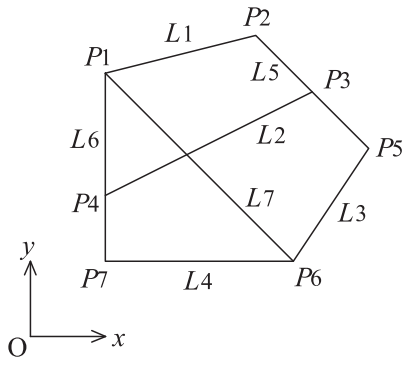


Figure 5. Example 2.

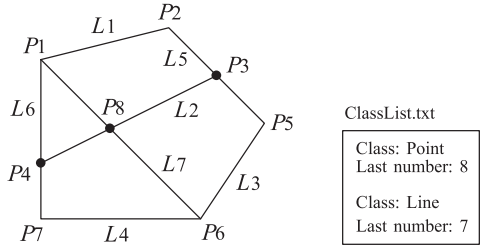


Figure 6. Intersections in Example 2.

(1) Search all intersections.
(2) Divide lines in their intersections.
(3) Calculate each angle between two lines.
(4) Search each of regions.

In Step (1) of this section, when the program made in Section 3 is executed, three intersections ($P3$, $P4$, $P8$) are recognized as in Fig. 6. In IFOG, the other points are not called intersections because they do not divide any lines. At this point, the states of ClassList.txt, Point.txt and Line.txt are illustrated in Fig. 7. In Step (2), four lines are divided into eight lines by the three intersections. For example, when $P3$ divides $L5$, the data of $L5$ is deleted and then two new lines ($L8$, $L9$) are added in Line.txt. Fig. 8 illustrates the drawing of this division. 9 illustrates updating text files in Fig. 8. In Point.txt of this figure, firstly three instances that are intersections are indicated and the property of "Divide lines" is added to each of them. For example, $L2$ and $L7$ are both divided by $P8$. So, "Divide lines: 2 7" is added as a new property to $P8$. Second, it is found that $P3$ is not an intersection after the division in updated Point.txt. Third, Class_Point.txt is updated and also Line.txt is updated. In this figure, for example, the variables of "Divide lines" are expressed such as "int t2 ... ". This expression means that there are several integer variables such as t2, t21, t22, etc. In Fig. 10, all divisions are executed. In this figure, $P4$ divides $L6$ into $L10$ and $L11$. Also, $P8$ divides $L2$ and $L7$ into $L12$, $L13$,

Point.txt

No.: 1
(x, y): 20 70
Kind:

No.: 2
(x, y): 60 80
Kind:

No.: 3
(x, y): 75 65
Kind: intersectin

No.: 4
(x, y): 20 37.5
Kind: intersectin

No.: 5
(x, y): 90 50
Kind:

No.: 6
(x, y): 70 20
Kind:

No.: 7
(x, y): 20 20
Kind:

No.: 8
(x, y): 42 48
Kind: intersectin

Line.txt

No.: 1
End points: 1 2

No.: 2
End points: 3 4

No.: 3
End points: 5 6

No.: 4
End points: 6 7

No.: 5
End points: 2 5

No.: 6
End points: 1 7

No.: 7
End points: 1 6

Figure 7. The text files in Fig. 6.

$L14$, $L15$. Fig. 11 illustrates updated text flies by the division. In Line.txt of this figure, $L2$, $L6$, $L7$ are deleted and $L10$, $L11$, $L12$, $L13$, $L14$, $L15$ are added. Also, ClassList.txt is updated in this figure.

In Step (3), each angle between two lines is calculated. In Fig. 12, three angles ($\theta 1$, $\theta 2$, $\theta 3$) about $L3$ are illustrated and three connected lines ($L9$, $L4$, $L15$) of $L3$ are emphasized to search a region. Their real values are indicated in updated Line.txt in Fig. 13. In this file, three properties are added that are "Length", "Connected lines" and "Angle". For example, "Connected lines at P 5" is a property of $L3$. This '5′ can be referred from "End points: 5 6". So, '9′ ($= L9$) becomes the value of the property.

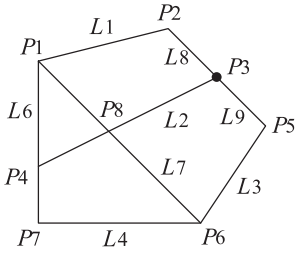


Figure 8. Division of $L5$ at $P3$ into two lines.

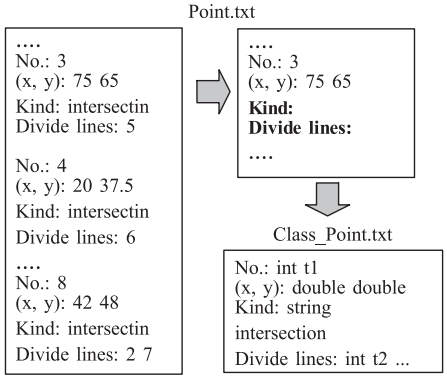


Figure 9. Updating text files in Fig. 8.

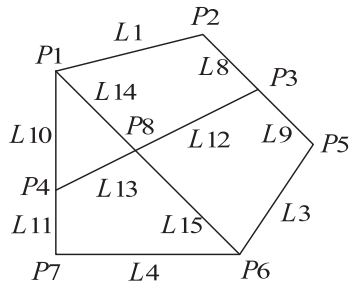


Figure 10. All divided lines by intersections.

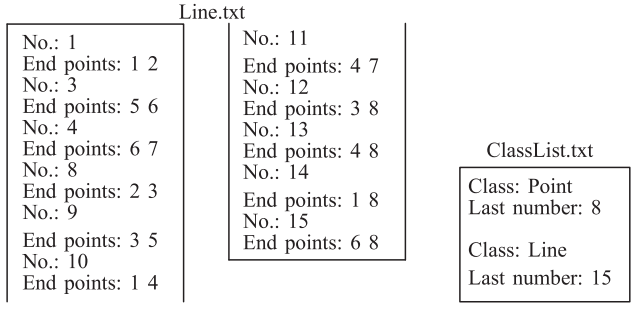
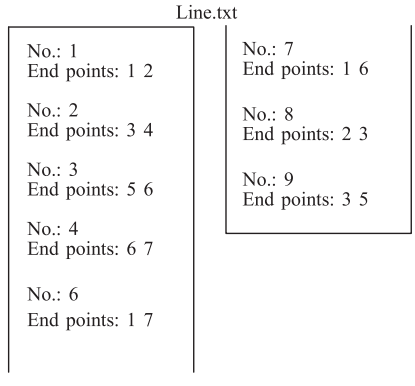


Figure 11. Updated text files in Fig. 10.



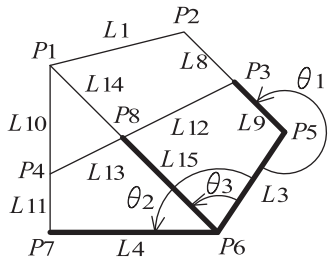


Figure 12. A region searched from *L*3.

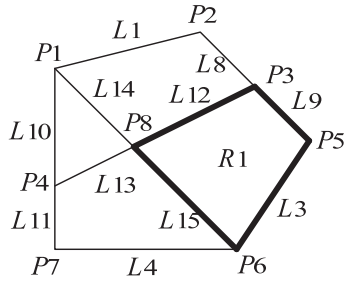


Figure 14. Searched region *R*1.

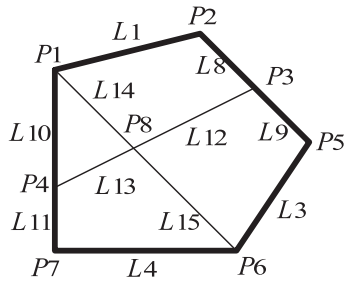


Figure 15. Searched closed line loop that is not a region.

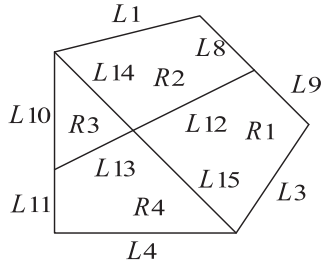


Figure 16. All regions.

Although there are two angles between two lines (*L*3, *L*9), the counterclockwise angle is always selected in this step. So, the angle from *L*3 to *L*9 is calculated as $\theta 1$, and the value of it is 258.69°. This value is the value of the property "Angle", and it is described under "Connected lines at P 5: 9". Also, since there are two angles ($\theta 2$, $\theta 3$) at *P*6 in *L*3, their values are described respectively under "Connected lines at P 6: 4 15". As the result, Line.txt is generalized as Class_Line.txt as in this figure.

In Step (4), each of regions is searched. In each region, there are no lines in its inside. To search a region, for example, when *L*3 is picked up firstly, the next line to make a region is searched at *P*5 or *P*6. If the line is searched at *P*6, *L*4 and *L*15 become its candidates. Since $\theta 3$ is smaller than $\theta 2$, *L*15 becomes the next line. In the same way, *L*12, *L*9 are selected continuously and then the region named *R*1 can be recognized as in Fig. 14. If the next line from *L*3 is searched at *P*5, *L*9 is selected and then *L*8, *L*1, *L*10, *L*11, *L*4 are selected continuously. As the result, a closed loop of lines is recognized as in Fig. 15. However, since there are four lines in the inside of the loop, it is found that this loop is not a region. Fig. 16 illustrates all regions (*R*1, *R*2, *R*3, *R*4) recognized from Example 2. Fig. 17 illustrates the text files about the regions. In this figure, firstly Class_Region.txt can be defined in Step (4) of Section 2. Here, "Elements" is a property whose values are line numbers. Also, "Kind" is a property whose values can be tetragon, triangle, polygon, rectangle, parallelogram, square, etc. These values can be obtained easily by using the lengths and angles of lines described in Line.txt as in Fig. 13. Then four instances of the class can be made as in Region.txt, and ClassList.txt is updated.

5. Discussion

IFOG would be effective for programming combinatorial searches of geometric elements such as regions in

Line.txt

```
....
No.: 3
End points: 5 6
Length: 3.6056
Connected lines at P 5: 9
Angle(°) from L 3 to L 9: 258.69
Connected lines at P 6: 4 15
Angle(°) from L 3 to L 4: 135
Angle(°) from L 3 to L 15: 78.69
....
```

Class_Line.txt

```
No.: int t1
End points: int t2 int t3
Length: double d1
Connected lines at P t2: int t4 ...
Angle(°) from L t1 to L t4: double d2
Connected lines at P t3: int t5 ...
Angle(°) from L t1 to L t5: double d3
```

Figure 13. Updated text files in Fig. 12.

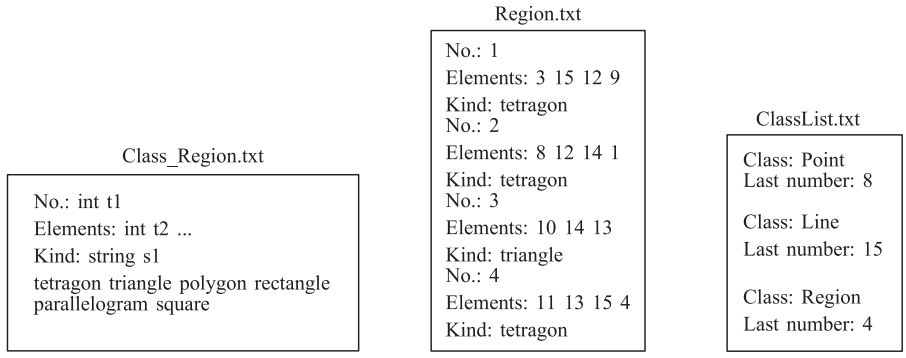


Figure 17. Text files about regions in Fig. 16.

Example 2. The recognition of the regions is fundamental for handling 2D drawings. Although only straight lines are handled in this paper, it would not be difficult to apply arcs and circles to IFOG. Initially, in DXF files, the data of each arc consists of center point(x, y), radius and two angles expressing its start point and end point. Also, the data of each circle consists of center point(x, y) and radius. Fig. 18 illustrates Example 3 that consists of straight lines, arcs and circles. When a region is recognized from $P1$ of $L1$, $L2$ and $L3$ become the candidates of the next line. If the direction of tangential line of $L2$ is different from the direction of $L3$, two angles of them to $L1$ can be compared. On the other hand, if the two angles are the same, a virtual straight line $P1P2$ can be compared with $L3$. As the result, a region can be recognized from $L1$, $L2$, $L5$ and $L4$ in Example 3.

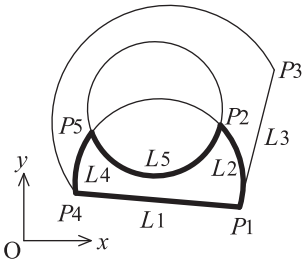


Figure 18. Example 3.

When more complex curved lines and 3D geometric elements are handled in IFOG system, the text files expressing their classes and instances would become large rapidly. It is an important issue for IFOG. However, basically reading and writing of the files are automatically executed by computers. Therefore, although it is important to make robuster format of the files, we estimate that the increase of the amount of information of the files does not become a serious problem. Therefore, high intelligent applications could be developed easily in 3D CAD systems by IFOG. In addition, since IFOG could always use minimum properties to develop programs, huge databases are not necessary to develop customized machine learning systems in geometric processing or in the other domains by applying IFOG techniques.

6. Conclusion

In this paper, IFOG (Inductive Functional prOgramming for Geometric processing) is proposed. In IFOG, geometric elements are expressed as their properties, and each property can be made from a programed function. The properties and functions could be knowledge for humans. Also, in IFOG, geometric elements are expressed as instances of their classes. Each class could be generalized from its instances, and each class and its instances are stored as text files in a PC. Therefore, programmers can always read and write the text files in their programing. So more flexible and smarter programming would be possible when they develop high intelligent applications in IFOG. The effectiveness of IFOG is indicated by using practical two examples in this paper, and it has been verified by our experimental system. Finally the issue and extensibility of IFOG is discussed in detail.

ORCID

Masaji Tanaka http://orcid.org/0000-0002-5266-9182
Yuki Takamiya http://orcid.org/0000-0002-8950-8833
Naoki Tsubota http://orcid.org/0000-0003-4826-5262

References

[1] Budd, T.: A Little Smalltalk, Addison-Wesley Publishing, 1987.
[2] Fujiwara, M.; Iwama, K.: A program that acquires how to execute sentences, WSEAS Transactions on Computers, 8(8), 2009, 1348–1357.
[3] Goldfarb, C. F.; Prescod, P.: XML Handbook (5th Edition), Prentice Hall PTR; 5 edition (December 8, 2003), 2003.
[4] Iwama, K.: A robotic program that acquires concepts and begins introspection, NueroQuantology, 4(4), 2006, 321–328.

[5] Tanaka, M.; Kaneeda, T.; Yamahira, T.; Iwama, K.: A Method to Restore Partial Omissions in 2D Drawings, Computer-Aided Design & Applications, 3(1–4), 2006, 341–347. http://dx.doi.org/10.1080/16864360.2006.10738472

[6] Tanaka, M.; Kaneeda, T.; Sasae, D.; Fukagawa, J.; Yokoi, R.: The Learning System to Restore Operations of Isolated Line Segments in 2D Drawings, Computer-Aided Design & Applications, 5(1–4), 2008, 354–362. http://dx.doi.org/10.3722/cadaps.2008.354-362

[7] Tanaka, M.; Kaneeda, T.: Feature Extraction from Sketches of Objects, Computer-Aided Design & Applications, 12(3), 2014, 300–309. http://dx.doi.org/10.1080/16864360.2014.981459

[8] Tanaka, M.; Iwama, K.; Hosada, A.; Watanabe, T.: Decomposition of a 2D Assembly Drawing into 3D Part Drawings, Computer-Aided Design, 30(1), 1998, 37–46. http://dx.doi.org/10.1016/S0010-4485(97)00051-1

[9] Wilson, P. R.; Wozny, M. J.; Pratt, M. J.: Geometric modeling for Product Realization, North Holland, 1993.