OBJECT ORIENTED GRAPHICS IN APL2

Manuel Alfonseca
IBM SofTec Lab
Paseo de la Castellana, 4
28046 Madrid (SPAIN)

Abstract

The paper describes a general and extensive graphic system in two or three dimensions, built in *APL2* by means of the object-oriented programming paradigm. The system fully demonstrates the appropriateness of the language to develop real-life applications using the most advanced techniques of computer science.

Introduction

In some previous papers (see references 1-5), the appropriateness of the *APL2* language for object-oriented programming (**OOP**) in general, has been discussed. This paper considers the special application of both *APL2* and **OOP** to a graphics environment.

Although this is not the place to include a full description of the **OOP** paradigm, the following paragraphs provide a very short introduction.

In current computer science, we can find three very different programming paradigms:

- *Procedural programming*, where the programs make up a hierarchy, while data are essentially appendices to the programs.
- *Logic programming*, where both programs and data are essentially unorganized.
- *Object-oriented programming*, where data and programs are encapsulated as **Objects**, which make up a hierarchy independent of the programs.

The basic elements and properties of **OOP** are (see references 6-8):

- **Objects**, a complex of data (properties) and programs (behavior) related to each other by means of relations. In the terminology of **OOP**, programs are called **methods**.
- **Classes**, or collections of objects related to the class they belong to by the **is-a** relation. Classes may be considered as special cases of objects, or as a completely different entity. The set of all classes forms a hierarchy.
- **Inheritance**, or the ability of an object to inherit properties and methods from the classes it belongs to.
- **Messages**, or requests from one object to another (possibly the same) object to activate a method.

Additional fundamental properties of **OOP** are the following:

- **Encapsulation**: all the information related to a given object (its properties) is hidden from other objects and is accessible only through the object behavior (its methods).
- **Polymorphism**: methods and properties may be made local to certain objects and their descendants. The same names can be used at different objects to perform related, but different procedures.
- **Dynamic binding**, or the ability to defer until execution time the decision on the program (method) to be executed in answer to a message. This usually means using an interpretive language, but a partial dynamic binding can also be attained with compiled languages by the use of certain techniques.

The graphic system

Graphic systems have always been one of the standard applications of object-oriented programming, and they are used frequently as easy-to-understand **OOP** examples.

The graphic system described here is written in *APL2/PC* (see reference 9), using the **OOP** exten-

sions described in reference 3 and the universal graphics processor (*AP207*), which is a part of the *APL2/PC* product.

A graphic object can belong to one of the following classes:

- Simple graphics (class name *SGRAPHIC*)
- Text graphics (class name *TGRAPHIC*)
- Composite graphics (class name *CGRAPHIC*)
- Chart graphics (class name *CHART*)

All of the preceding object classes are subclasses of a single class called *GRAPHIC*. Each of them has a single method defined which, in all cases, is called *DRAW* (polymorphism), the function of which is to draw the corresponding graphic on the screen, using *AP207*.

To each of these methods corresponds a different *APL2* function, whose name is the catenation of the subclass name, to an under-bar, to the method name, according to the rules used in the *APL2*-**OOP** system. For example, the version of method *DRAW* applicable to class *SGRAPHIC* will be embodied in function *SGRAPHIC_DRAW*.

Inheritance is also useful to build a family of related graphics that differ only in specific properties. They can be made descendants of a single object, where all the common properties are defined, while each object in the family has to define only its own specific properties.

Simple graphics

A simple graphic (class name *SGRAPHIC*) is the description of an elementary drawing consisting of straight lines. For example, a square can be described with the following properties:

```
VIS   0  1  ¯1  ¯1  1
X     1  ¯1 ¯1  1   1
Y     1  ¯1 ¯1  1   1
```

This object is represented as a two column general array. The first column contains the property names, the second the property values. The value of a property (*Y*, for instance) can be a vector (`1 ¯1 ¯1 1 1`, as in the example).

The property *VIS* (visibility) controls the movement of the *pen* on the screen. A value of 0 indicates that the pen must move, without leaving any trace, to the point whose coordinates are the corresponding values of the properties *X* and *Y*. A value of 1 indicates that a visible line must be drawn from the present position to the indicated point.

Of course, the fact that the object belongs to a hierarchy, implies that this is not the only information it contains: additional rows must be added to the two column matrix. This is done automatically if the object is created using the APL2/OOP functions described in reference 3:

```
MESSAGE 'SGRAPHIC' 'CREATE' 'SQUARE'
ASSIGN 'SQUARE' 'VIS' (0 1 1 1 1)
ASSIGN 'SQUARE' 'X' (1 1 ¯1 ¯1 1)
ASSIGN 'SQUARE' 'Y' (1 ¯1 ¯1 1 1)
```

where *SQUARE* is the name of the object containing the definition of the simple graphic (which can be either an *APL2* variable or a record in an *AP211* file, depending on the position of object *GRAPHIC* in the **OOP** hierarchy), and *MESSAGE* and *ASSIGN* are two of the *APL2*-**OOP** functions described in reference 3. The arguments of *MESSAGE* are: object, method and parameters. The arguments of *ASSIGN* are: object, property and value.

Additional properties for simple graphics are:

- *Z*: the third coordinate, for three-dimensional graphics. Its default value is zero.
- *COLOR*: the color in which each straight segment will be drawn. A value of zero represents the default color (white). Other positive integers represent the different colors, according to the outstanding palette.
- *PATTERN*: a value of zero (the default) indicates that the drawing is *transparent* (or wiry). Other positive integer values indicate that the lines being drawn make a closed polygon that should be filled with the indicated pattern.
- *STYLE*: a value of zero (the default) indicates that the corresponding segment must be drawn in full. Other positive integer values select different line styles, like **dashed**, **dotted**, **broken**, and so forth.

All the properties of a simple graphic should have the same number of elements, or be absent, in which case the default value is applied to every segment.

Text graphics

A text graphic (class name *TGRAPHIC*) consists of nothing but texts. The following is an example:

```
TEXT    TITLE  TEXT1  TEXT2  TEXT3  TEXT4
X       ¯2.4   ¯7     4.6    ¯7     4.6
Y        7      2      2     ¯3     ¯3
SCX      2      1      1      1      1
SCY      2      1      1      1      1
COLOR   12     14     14     14     14
```

Text graphics are represented in the same way as simple objects, by a two column general array. The preceding object would be created in the following way:

```
MESSAGE 'TGRAPHIC' 'CREATE' 'SAMPLE_TEXT'
ASSIGN 'SAMPLE_TEXT' 'TEXT' ('TITLE' 'TEXT1'
      'TEXT2' 'TEXT3' 'TEXT4')
ASSIGN 'SAMPLE_TEXT' 'X' (¯2.4 ¯7 4.6 ¯7 4.6)
ASSIGN 'SAMPLE_TEXT' 'Y' (7 2 2 ¯3 ¯3)
ASSIGN 'SAMPLE_TEXT' 'SCX' (2 1 1 1 1)
ASSIGN 'SAMPLE_TEXT' 'SCY' (2 1 1 1 1)
ASSIGN 'SAMPLE_TEXT' 'COLOR' (12 14 14 14 14)
```

The properties *X* and *Y* give the position of the lower hand corner of each text; *SCX* and *SCY* are size multipliers for each axis; and *COLOR* defines the color in which each text will be drawn, as in the simple graphics.

Additional properties for text graphics are:

- *Z*: the third coordinate, for three-dimensional graphics. Its default value is zero.
- *ANG*: the angle in degrees of the rotation to be applied to the text around the *Z* axis. This makes it possible to draw texts on the screen in any direction. Positive and negative values correspond to counterclockwise and clockwise rotations, respectively.
- *WIDTH*: a value of zero (the default) indicates that the text will be drawn with narrow characters. Other positive integer values (1-99) indicate that wider characters should be used. The values of `10 10 ⊤ WIDTH` indicate the width along the *Y* and the *X* axis, respectively.
- *STYLE*: a value of zero (the default) indicates that the default font (Roman) should be used to draw the text. Other positive integer values select different fonts, like **italics**, and so forth.
- *DEV* (deviation): a value of zero indicates that the font definition should be used as it is. Other integer values indicate that a deviation should be applied to each character in the font. Positive values select deviations to the right, negative values to the left. Using deviations, a Roman font can be converted into italics.

All the properties of a text graphic should have the same number of elements, or be absent, in which case the default value is applied to every text.

Composite graphics

Composite graphics (class name *CGRAPHIC*) contain references to other graphics of any type, together with geometrical and other transformations to be applied to the referenced graphics. Since the latter can also be composite, the definition of a composite graphic is recursive. The following is an example of a composite graphic:

IMAGE	*TDIA3*	*AXIS*	*SQUARE*	*SQUARE*	*SQUARE*
X	0	0	¯5	¯0	3
Y	0	0	¯5	¯1	3.82
SCX	1	1	2	3	2
SCY	1	1	2	2	2
ANZ	0	0	0	0	45
COLOR	14	0	10	12	11

Again, composite graphics are represented by a two column general array. The preceding object would be created in the following way:

```
MESSAGE 'CGRAPHIC' 'CREATE' 'DIA3'
ASSIGN 'DIA3' 'IMAGE' ('AXIS' 'SQUARE'
      'SQUARE' 'SQUARE')
ASSIGN 'DIA3' 'X' (0 0 ¯5 0 3)
ASSIGN 'DIA3' 'Y' (0 0 ¯5 ¯1 3.82)
ASSIGN 'DIA3' 'SCX' (1 1 2 3 2)
ASSIGN 'DIA3' 'SCY' (1 1 2 2 2)
ASSIGN 'DIA3' 'ANZ' (0 0 0 0 45)
ASSIGN 'DIA3' 'COLOR' (14 0 10 12 11)
```

Property *IMAGE* is the name of another graphic object (simple, text or composite); *X* and *Y* define a translation; *SCX* and *SCY* a scale change; *ANZ* a rotation around the *Z* axis; and *COLOR* a color change to be applied to each image.

Additional properties for composite graphics are:

- *Z*: the translation along the third coordinate, for three-dimensional graphics. Its default value is zero.
- *SCZ*: the scale change along the third coordinate, for three-dimensional graphics. Its default value is 1.
- *ANX*: the angle in degrees of the rotation to be applied to the image around the *X* axis. It is used in three-dimensional graphics. Positive and negative values correspond to counterclockwise and clockwise rotations, respectively.
- *ANY*: the angle in degrees of the rotation to be applied to the image around the *Y* axis. It is used in three-dimensional graphics. Positive and negative values correspond to counterclockwise and clockwise rotations, respectively.
- *PATTERN*: an overriding pattern value to be applied to the corresponding (simple or composite) image. A value of zero (the default) indicates that the *PATTERN* property of the corresponding image is in effect.
- *STYLE*: an overriding style value to be applied to the corresponding image. If the corresponding image is a text graphic, this parameter is assumed to select a font, if it is a simple graphic, it represents a line style. A value of zero (the default) indicates that the *STYLE* property of the corresponding image is in effect.

All the properties of a composite graphic should have the same number of elements, or be absent, in which case the default value is applied to every text.

In the example given above, the following images are invoked:

- *SQUARE* is the same as the simple graphic example given above.
- *TDIA3* is the following text graphic:

```
TEXT   COMPOSITE SLIDE
X       ¯4.4
Y        7
```

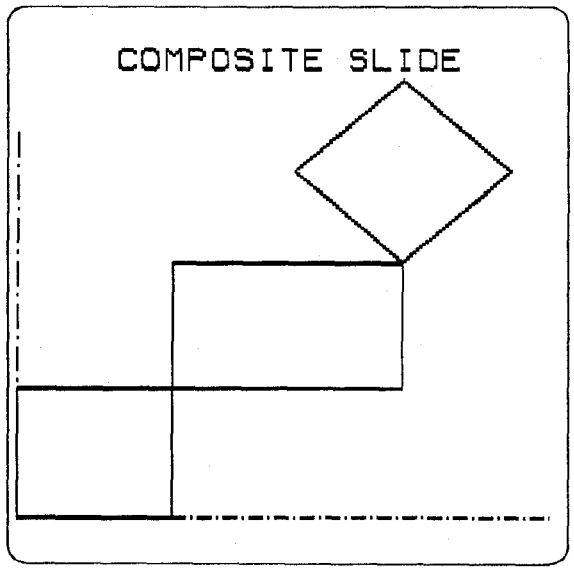
- *AXIS* is the following simple graphic:

```
VIS     0  1  0  1
X      ¯7 ¯7 ¯3 ¯7
Y       5 ¯3 ¯7 ¯7
STYLE   3  3  3  3
```

The screen image corresponding to the composite graphic *DIA3* is obtained by executing:

```
MESSAGE 'DIA3' 'DRAW'
```

The result of the execution of this message is shown in the following figure:



Three-dimensional graphics

From the above description, it will be evident how the preceding graphic classes are sufficient to support a three-dimensional graphic system. However, an additional property is required, namely the perspective procedure to be used while drawing the images, which should affect the whole system, not only each individual object. Therefore, this property should be defined at class *GRAPHIC*, the root of the graphic system.

ΔPERSPEC is a function that has been provided to allow the interactive selection of the appropriate perspective system for three-dimensional images. The following perspective systems are supported:

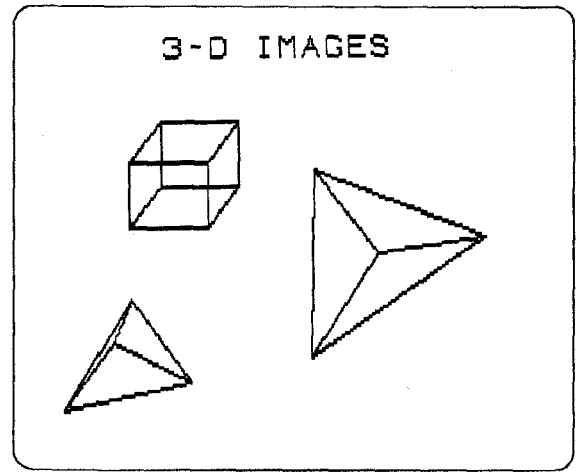
- Isometric Axonometric
- Cavalier
- Conic over Z = 0
- Conic over vertical glass
- Conic over oriented glass
- General perspective definition

The following are two examples of images obtained using the three-dimensional capabilities of the *APL2*-**OOP** graphic system:

- Image *DIA17*: It is defined as follows:

```
IMAGE  TDIA17  CUBE  TETRA  TETRA
X          0    ¯3     ¯4     ¯3
Y          0     2     ¯4     ¯1
SCX        1     2    2.5      4
SCY        1     2    2.5      4
SCZ        1     2    ¯2.5     4
ANZ        0     0     ¯5     30
COLOR     14    13     10     14
```

and drawn on the screen as in the following figure:



- Image *ESP29* represents an opaque icosahedron, which is drawn as indicated by the following figure:

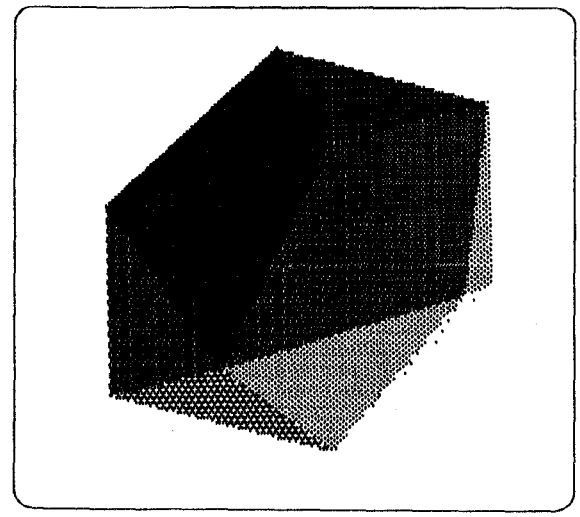


Chart graphics

Chart graphics (class name *CHART*) are simple time series that can be represented on the screen as a chart of different types. For example, the following object is called *CHART1* and is an instance of class *CHART*:

```
LIST  10       20       30       40
TEXT  Case 1   Case 2   Case 3   Case 4
TITLE SAMPLE CHART  Just for Demo
```

An additional property, *COLOR*, allows the programmer to select the desired colors for the charts.

To obtain a line chart of object *CHART1*, it is enough to execute the following message:

```
MESSAGE 'CHART1' 'DRAW'
```

However, this message has one parameter that allows us to change the type of chart we want to obtain, either to a vertical-bar-chart or to a pie-chart, as in the following examples:

```
MESSAGE 'CHART1' 'DRAW' 'VBAR'
MESSAGE 'CHART1' 'DRAW' 'PIE'
```

The corresponding images can be seen in the next page.

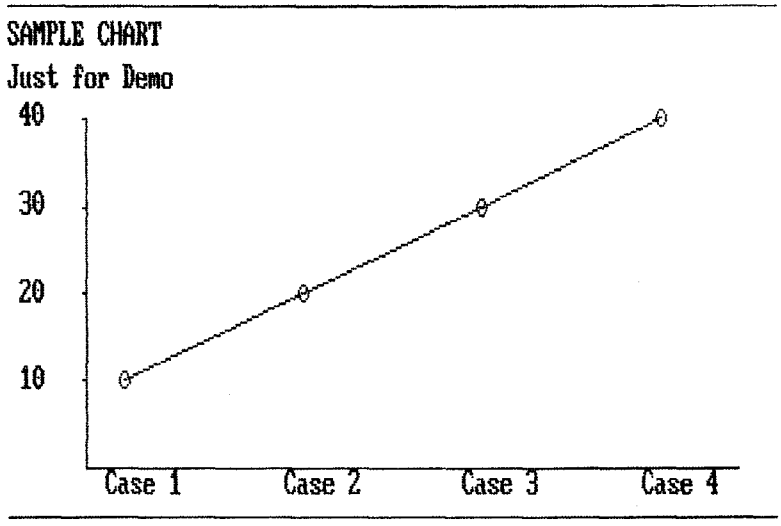
Conclusion

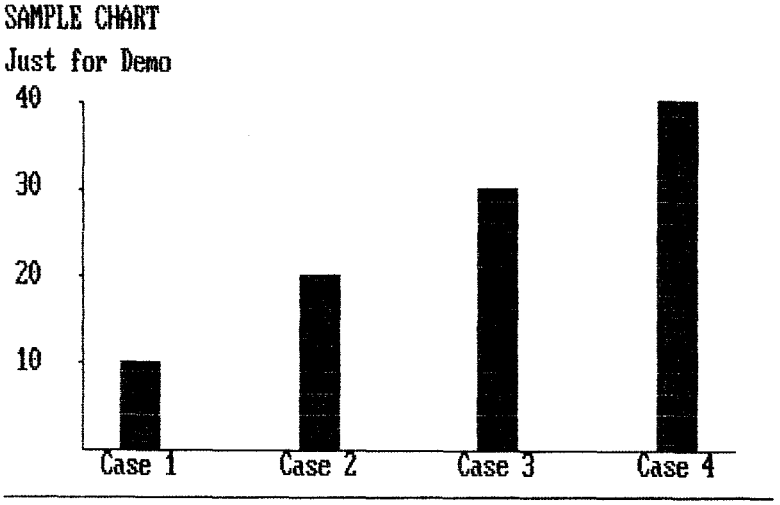
The usefulness of *APL2* for object-oriented programming has been proved before (see references). This paper strengthens the case by describing the implementation of a classical **OOP** application: graphics. The graphic system described in the paper is very simple (there are only four subclasses of graphic objects), but also extremely general (it even supports three-dimensional graphics in many perspective systems) and should be considered as a real application, which has been used to produce many slides for presentations. Using the *APL2*-**OOP** routines, the graphics can even be kept in a data base, and created and modified very easily. The **OOP** environment makes further extensions to the system natural and straightforward.

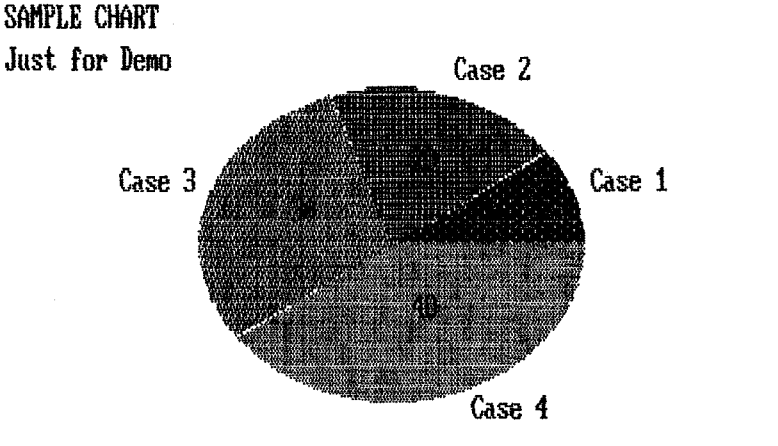
The application had been previously developed in plain APL. The introduction of the **OOP** paradigm made the design much easier, simplified the code (through straightforward reuse of the APL2-OOP support described in reference 3) and made testing and debugging simpler. Of course, something had to be paid as regards performance, but the loss is negligible for this kind of applications, and perfectly acceptable.

References

1. Alfonseca, M. *Object Oriented Programming in APL2* APL Quote Quad, Vol. 19, No. 4, p.6-11, 1989.
2. Gfeller, M. *Object Oriented Programming in AIDA APL* APL Quote Quad, Vol. 19, No. 4, p.164-168, 1989.
3. Alfonseca, M. *Frames, Semantic Networks and Object-Oriented Programming in APL2* IBM Journal of Research and Development, 33:5, p. 502-510, Sep. 1989.
4. Alfonseca, M. *Object Oriented Programming Tutorial* APL Quote Quad, Vol. 20, No. 4, p.9, 1990.
5. Pantziarka, P. *Object Oriented Database using Frames in Second Generation APL* APL Quote Quad, Vol. 20, No. 4, p.284-287, 1990.
6. Cox, B. *Object-Oriented Programming: an Evolutionary Approach.* Addison Wesley, 1986.
7. Meyer, B. *Object-Oriented Software Construction.* Prentice Hall, 1988.
8. *Smalltalk-V User's Guide.* Digitalk Inc.
9. *APL2 for the IBM Personal Computer*, Program Number 5799-PGG, Part No. 6242036, IBM Corporation.


SAMPLE CHART
Just for Demo
40
30
20
10
Case 1
Case 2
Case 3
Case 4


SAMPLE CHART
Just for Demo
40
30
20
10
Case 1
Case 2
Case 3
Case 4


SAMPLE CHART
Just for Demo
Case 2
Case 3
Case 1
Case 4