ABSTRACT

Software is expensive in that it costs money to acquire and time to install and support. This paper presents a simple turtle graphics system, implemented in Excel Basic, that can be used in any environment that already supports Excel spreadsheets. The system can be used as a black-box module in object form for beginning courses to introduce students to programming via graphics or in source form for advanced courses to teach modular interface design and graphics in general. An example is presented that generates a fractal curve. This paper would be of interest to anyone teaching programming in general and Excel Basic in particular.

INTRODUCTION

Turtle graphics is a concept made popular by [1], an excellent reference for ideas on programming using turtle graphics, and implemented in the LOGO programming system. This paper develops a minimal subset of turtle graphics capability, leaving extensions as exercises. Using a standard spreadsheet allows the graphics output to be handled in the same was as any other chart object (e.g., copy-paste into a document). The module development is also an excellent introduction to what is here called Excel Basic [2], to avoid confusion with the Visual Basic that is used to create stand-alone applications programs. Although the graphics are not interactive graphics, but static, once the graphics are generated on a canvas, the drawing tools of the spreadsheet can be used interactively to "touch up" the graphics.

The target audience for this turtle graphics system consists of students who have previous experience with a spreadsheet (e.g., worksheets and charts), but who have never before written a computer program. Turtle graphics provides a nice, convenient, and non-threatening way for such students to be introduced to programming. However, there are many opportunities to extend the described system in various ways for students who are familiar with programming.

A salient feature of the implemented system is its simplicity. Most of the nitty-gritty details of the functionality have already been implemented in the spreadsheet program. What remains to be done is to add a module wrapper to allow access to that functionality. And this, in turn, becomes an excellent example of modular interface
design. There is indeed an underlying semantics to the module. For example, should the initial pen position be up or down? Many students do not realize that in some problems the interface is set and one must adapt the program to the interface. In other problems, the program is set and one must adapt the interface to the program. But, for the current problem, one has the flexibility to design the interface and the program together. The goal, therefore, is to develop a good interface.

The code is now discussed in the order in which the statements appear in the program, providing a short introduction to Excel Basic for those unfamiliar with the system.

THE TURTLE

The turtle moves on a canvas, leaving marks on the canvas if the pen is down. The pen position can be represented as an integer variable, private to the module.

**Private penpos As Integer**

Constants are used to name the positions.

**Private Const pendown As Integer = 0**
**Private Const penup As Integer = 1**

The module interface includes commands (i.e., procedures) to raise the pen and to lower the pen.

**Public Sub up**
penpos = penup
End Sub

**Public Sub down**
penpos = pendown
End Sub

Note that, since the pen is implemented in software and not hardware, a conditional statement is not used to insure that the pen position is only changed if the request is for a change in position. This makes the code simpler.

The position of the turtle can be represented with double precision (i.e., floating point) numbers as follows.

**Private xpos As Double, ypos As Double**

A crucial element of the design is that, in the existing system, the upper-left corner is the position \((x=0, y=0)\) where \(x\) is the horizontal dimension (positive to the right) and \(y\) is the vertical dimension (positive going down). Exercise: Modify the final module so that the lower-left corner is the position \((x=0, y=0)\). The maximum values in each direction depend on the current video mode (e.g., VGA, SVGA, etc.) and the adaptation of the code to the current mode is left as an exercise.

The turtle can be moved to position \((x, y)\) by setting \(xpos\) to \(x\) and setting \(ypos\) to \(y\). However, if the pen is down, a line should be drawn.
Public Sub move(x As Double, y As Double)
    If penpos = pendown Then
        addline xpos, ypos, x, y
    End If
    xpos = x
    ypos = y
End Sub

The procedure to draw a line can be implemented as follows.

Private Sub addline(_
    x1 As Double, y1 As Double, _
    x2 As Double, y2 As Double)
    With ActiveSheet.Lines.Add(x1, y1, x2, y2)
        ' ... set desired line properties
    End With
End Sub

Note that this procedure is private to the module. Also, the underscore character
"_" is used to continue a statement on the next line and the quote character "'" is used
to introduce a comment. The creation of a relative move command is left as an exercise.
As with other Excel Basic commands, the easiest way to find the relevant command is
to record a macro that uses the command, inspect the generated code, and use the help
system to find additional information on that particular command.

THE CANVAS

All drawing takes place on the canvas, which is initially blank. The canvas could
be on a spreadsheet, on a dialog box, or on a chart. For the present purposes, a chart
area of a chart is selected. Modifying the code for either spreadsheet or dialog box is
left as an exercise. The trickiest part of the design is the initialization of the canvas,
since this part of the code must access the underlying spreadsheet object system in a
very specific manner. One way to do the graphics initialization is as follows.

Public Sub initturtle(s As String)
Dim i As Integer
    penpos = penup
    xpos = 0
    ypos = 0
If s = "" Then
    Set c = Charts.Add
Else
    I = sheetpos(s)
    If i = 0 Then
        Set c = Charts.Add
        ActiveSheet.Name = s
    ElseIf TypeName(Sheets(i)) = "Chart" Then
        Sheets(i).Select
        Set c = ActiveSheet
    Else
        Set c = Charts.Add
    End If
End If

C.ChartArea.Clear

For i = C.Lines.Count To 1 Step -1
    C.Lines(i).Delete
Next

End Sub

If no chart name is supplied, a new chart is created. If the supplied chart name exists and that name represents a chart, that existing chart is used. If the supplied chart name does not exist, a new chart with that name is created. Otherwise, a new chart is created. Note the use of the "set" keyword to indicate a object assignment that is similar to a pointer operation in other imperative languages. The canvas is then cleared.

The function to check existing for the index of an existing sheet, a function useful for other purposes, is as follows.

Private Function sheetpos(s As String) As Integer
Dim i As Integer, j As Integer

    j = 0
    For i = ActiveWorkbook.Sheets.Count To 1 Step -1
        If UCase(Sheets(i).Name) = UCase(s) Then
            j = i
        End If
    Next

    sheetpos = j
End Function
MODULE INTERFACE

The entire module can be easily packaged as an "Add-In", effectively compiling the source code (i.e., a file of type .xls) into object code (i.e., a file of type .xla). The resulting object code is loaded into the spreadsheet in the same way as source code. The functionality is available as a black box module with the following module interface.

Public Sub initturtle(s As String)
Public Sub up()
Public Sub down()
Public Sub move(x As Double, y As Double)

The beauty of using Excel Basic is that all of the programming features of the language are available, in addition to the added functionality provided by the turtle graphics module.

Assuming that the "Add-in" is saved as file turtles.xla, one way to use the "Add-in" from Excel and start a program is as follows.

- Select "File", "Open", turtles.xla, "OK".
- Select "Insert", "Macro", "Module", to start a new module file in the current workbook.
- Select "Tools", "References", scroll to the bottom of the list box, and insure that "TURTLES.XLA" is checked. (Note: In Excel 97, one needs to open the turtles.xls add-in and drag the module to the project for the current spreadsheet so that the proper references are established). Otherwise, the module being created will not have access to the turtle graphics module. Select "OK".

FRACTAL CURVES

An example of the module in action is in the generation of fractal curves [3]. Although there are many commercial and public domain programs available to create fractals, some of the essential ideas of fractals can be learned using turtle graphics. And for the beginning student, the personal creation of a fractal curve is very satisfying. One of the nice features of fractal curves is that they can be generated recursively from a base and step case without the need for looping actions. This is important if beginning students are being introduced to programming, since the looping action in an imperative language is a fairly difficult concept to grasp effectively. In addition, the use of fractal curves allows the generation of a lot of impressive output without writing a lot of code. The Koch coastline fractal curve is used for example purposes.
The Koch coastline of degree 0, the base case, appears as follows.

\[(x_0, y_0) \quad \rightarrow \quad (x_4, y_4)\]

The coordinates \((x_0, y_0)\) and \((x_4, y_4)\) are the start and end points, respectively, of the base case.

The Koch coastline of degree 1, the step case, appears as follows.

\[(x_0, y_0) \quad \rightarrow \quad (x_1, y_1) \quad \rightarrow \quad (x_2, y_2) \quad \rightarrow \quad (x_3, y_3) \quad \rightarrow \quad (x_4, y_4)\]

The coordinates \((x_0, y_0)\), \((x_1, y_1)\), \((x_2, y_2)\), \((x_3, y_3)\), and \((x_4, y_4)\) form the step case.

The module code to generate a Koch coastline of degree 3 that starts at point \((40,100)\) and ends at point \((640,100)\) is as follows.

These two cases, along with some vector arithmetic and algebra, are enough to write the following procedure.

```vba
Sub coastline(ByVal n As Integer, _
    ByVal x0 As Double, ByVal y0 As Double, _
    ByVal x4 As Double, ByVal y4 As Double) _
Dim x1 As Double, y1 As Double _
Dim x2 As Double, y2 As Double _
Dim x3 As Double, y3 As Double
```
If \( n = 0 \) Then
  up
  move \( x_0, y_0 \)
  down
  move \( x_4, y_4 \)
  up
Else
  \( x_1 = x_0 + (x_4 - x_0) / 3 \#
  y_1 = y_0 + (y_4 - y_0) / 3 \#
  x_3 = x_0 + 2 \# \* (x_4 - x_0) / 3 \#
  y_3 = y_0 + 2 \# \* (y_4 - y_0) / 3 \#
  x_2 = (x_1 + x_3 + y_1 - y_3) / 2 \#
  y_2 = (y_1 + y_3 + x_3 - x_1) / 2 \#
  coastline n - 1, x_0, y_0, x_1, y_1
  coastline n - 1, x_1, y_1, x_2, y_2
  coastline n - 1, x_2, y_2, x_3, y_3
  coastline n - 1, x_3, y_3, x_4, y_4
End If
End Sub

Note the following.

- The procedure coastline draws a Koch coastline of degree \( n \) from point \((x_0, y_0)\) to point \((x_4, y_4)\).
- A double precision literal such as 2.0 is written as 2#.
- A value parameter is specified using the ByVal keyword (not strictly necessary here).
- Vector arithmetic is used to determine points \((x_1, y_1)\), \((x_2, y_2)\), and \((x_3, y_3)\), the exact formulation of which is left as an exercise.
- As an exercise, modify the code to, instead of always placing point \((x_2, y_2)\) on the same side of the curve, randomly pick a side. This produces a much more realistic coastline.
- An additional exercise is to modify the code to avoid redundant up and down pen movements. Hint: One way requires an outer procedure to initially move the pen and then, reasoning via a structural induction, determine that the inner recursive procedure need not move the pen up and down, or even move to the start position.
A Koch coastline of degree 3 would appear as follows.

The module code to generate a Koch coastline of degree 3 that starts at point (40,100) and ends at point (640,100) is as follows.

Sub dokoch()
  initturtle "Koch#03"
  coastline 3, 40, 100, 640, 100
End Sub

ADDITIONAL EXERCISES

The following are some useful exercises for advanced programming classes that involve modifying the source code to the module in degree to increase functionality and/or robustness.

- Implement the more traditional turtle graphics commands, including pendown and penup (this requires renaming within the module), xcor and ycor (as functions that act as properties), and the movement actions of back, forward, left, right, and face.

- In order to demonstrate that the proper viewpoint can make a big difference in both the ease of writing correct programs and the clarity of program code, rewrite the above fractal coastline program to use only local turtle movement commands such as right, left, and forward instead of the absolute move command.

- Use a structural induction argument to move the current up and down pen movements to another location in the program so that, during program execution, only one up and one down pen movement is executed.

- Implement traditional graphics primitives of moveto, rmoveto, lineto, and rlineto in terms of up, down, and move.
• Implement viewport windows and transformations that include translation, rotation, and scaling. Include a general transformation matrix in addition to separate procedures.

• Add the facility for additional shapes supported by the interactive drawing toolbar, such as rectangle, ellipse, arc, freeform, textbox, arrow, freehand, and fill operations.

• Using the underlying object system, "reverse engineer" the generated graphics in order to generate the list of instructions necessary to create the graphics. Output these in a standard graphics format, such as an encapsulated PostScript file (i.e., with a bounding box).

• Extend the module to an object, converting variables (e.g., the position of the pen) into properties and procedures (e.g., to move the turtle) into methods.

• Add error checking and recovery actions to the initturtle command so that runtime errors are minimized and/or avoided entirely. For example, taking appropriate action if the desired name is already in use by a non-chart object. Or, adding a hidden label on a chart to identify it as one created by the turtle graphics module and not by some other means, so that an existing chart is not inadvertently overwritten.

SUMMARY

This paper has presented a minimal subset of turtle graphics that can be easily implemented on a readily accessible spreadsheet program. Such a system can be used to introduce students to programming. An example was presented that used fractal curves as an application of the turtle graphics module.

REFERENCES

