

TegoSoft Inc.

Copyright © 1994-1996 TegoSoft Inc. ® All Rights Reserved

P.O.Box 389, Bellmore, NY 11710

Web Site: <http://www.tegosoft.com>

 e-mail: tegosoft@msn.com

Lesson 23 - Free-Hand (Pencil) Drawing


In this lesson you'll learn to write programs that draw like the Pencil tool of Paint (or Paintbrush).


Creating the Directory of the Project and Saving the Files of the Project


As usual, you'll start by creating the directory of the project, and saving the files of the project.

 Create the **C:\VBMyProg\Lesson23** directory.


You'll save the files of this lesson to this directory.

 Select **New Project** from the **File** menu of Visual Basic.

 Make the window of Form1 the selected window, select **Save File As** from the **File** menu of Visual Basic, and save the file as **MyPencil.FRM** inside the **C:\VBMyProg\Lesson23** directory.

 Select **Save Project As** from the **File** menu of Visual Basic, and save the project as **MyPencil.VBP** inside the **C:\VBMyProg\Lesson23** directory.

Always Use Option Explicit

 Inside the general declarations section of Form1 type the following code:

```
' Force variable declarations  
Option Explicit
```

The **Option Explicit** statement forces you to declare variables before using the variables.

Designing the Window of the MyPencil Program

You'll now design the window of the MyPencil program so that it will look as shown in Figure 23.1.

 Set the properties of Form1 as follows:

Name: Form1


Caption: The MyPencil Program

Icon: Set the Icon property to the file **Pencil04.ICO** that resides inside the **\Icons\Writing** directory of Visual Basic.

BackColor: White

Implementing the Exit Button

You'll now implement the Exit button:

 Place a CommandButton inside Form1, then set the properties of the CommandButton as follows:

Name: cmdExit

Caption: E&xit

Your Form1 should now look as shown in Figure 23.1

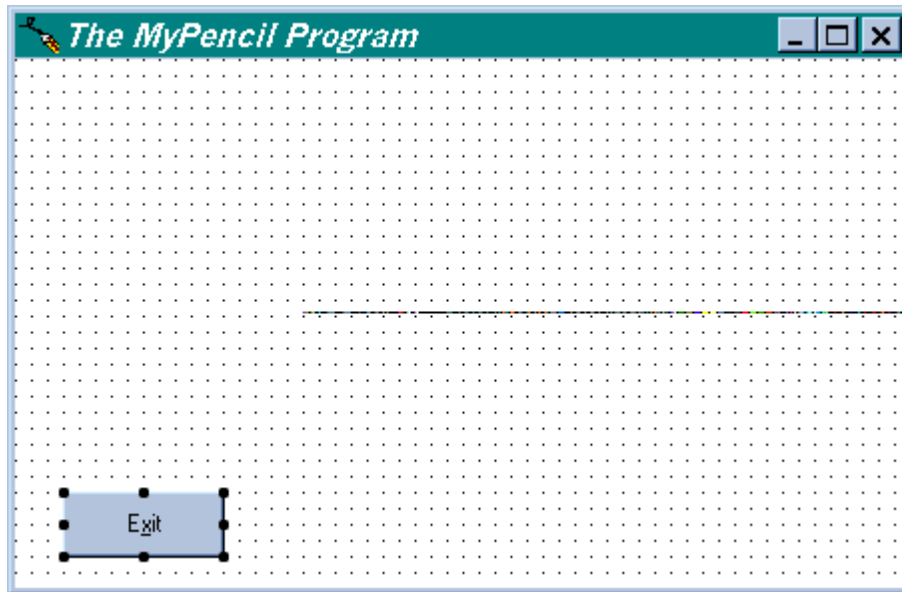



Figure 23.1. Form1 of the MyPencil program in design mode.

 Add the following code inside the **cmdExit_Click()** procedure:

```
Private Sub cmdExit_Click()
```


```
End
```

```
End Sub
```

So whenever the user clicks the Exit button, the MyPencil program terminates itself.

Attaching Code to the MouseMove Event of Form1

You'll now type code inside the **Form_MouseMove()** procedure:

 Double click Form1 to display the Code window, set the **Object** list box to Form1, and set the **Proc** list box to **MouseMove**.


Visual Basic responds by displaying the **Form_MouseMove()** procedure ready to be edited by you. Add the following code inside the **Form_MouseMove()** procedure:


```
Private Sub Form_MouseMove(Button As Integer, _  
                            Shift As Integer, _  
                            X As Single, _  
                            Y As Single)
```

```
    If Button = 1 Then  
        Beep  
    End If
```

```
End Sub
```


The **Button** parameter of the **Form_MouseMove()** is updated automatically by the program. If the user presses the left button of the mouse while moving the mouse, the **Button** parameter is set to **1**.

 Select **Save Project** from the **File** menu of Visual Basic to save your work. Execute the MyPencil program.

 Press the left button of the mouse, and while keeping the left button of the mouse down, move the mouse.

As you move the mouse, you hear the PC beeps.

=====

 Experiment with the MyPencil program, then click the Exit button to terminate the program.

The point of the preceding experiment is to realize that as long as you move the mouse, the **MouseMove** event occurs, and therefore the **Form_MouseMove()** procedure is executed. It is important to understand that the PC periodically checks the status of the mouse. If the PC discovers that the mouse was moved since the last time the mouse was checked, the PC causes the program to generate the **MouseMove** event.

Drawing Points According to the Mouse Movements

You'll now replace the Beep statement that you typed inside the **Form_MouseMove()** procedure with code that draws points according to the mouse movements:

 Modify the code inside the **Form_MouseMove()** procedure so that it will look as follows:

```
Private Sub Form_MouseMove(Button As Integer, _  
                           Shift As Integer, _  
                           X As Single, _  
                           Y As Single)
```

```
    If Button = 1 Then
```

```
        PSet (X, Y)
```

```
    End If
```


```
End Sub
```

The code that now appears under the **If** statement draws a point at coordinates (X,Y). **X** and **Y** are parameters of the **Form_MouseMove()** procedure. **X** and **Y** represent the coordinates of the mouse cursor at the time the **Form_MouseMove()** procedure is executed.


Let's see your code in action:

 Select **Save Project** from the **File** menu of Visual Basic to save your work.

 Execute the MyPencil program.

 Place the mouse cursor inside the window of the MyPencil program, press down the left button of the mouse, and while the left button of the mouse is pressed down, move the mouse.

The MyPencil program responds by drawing points according to the mouse movement (see Figure 23.2).

 Experiment with the MyPencil program and then click the Exit button to terminate the program.

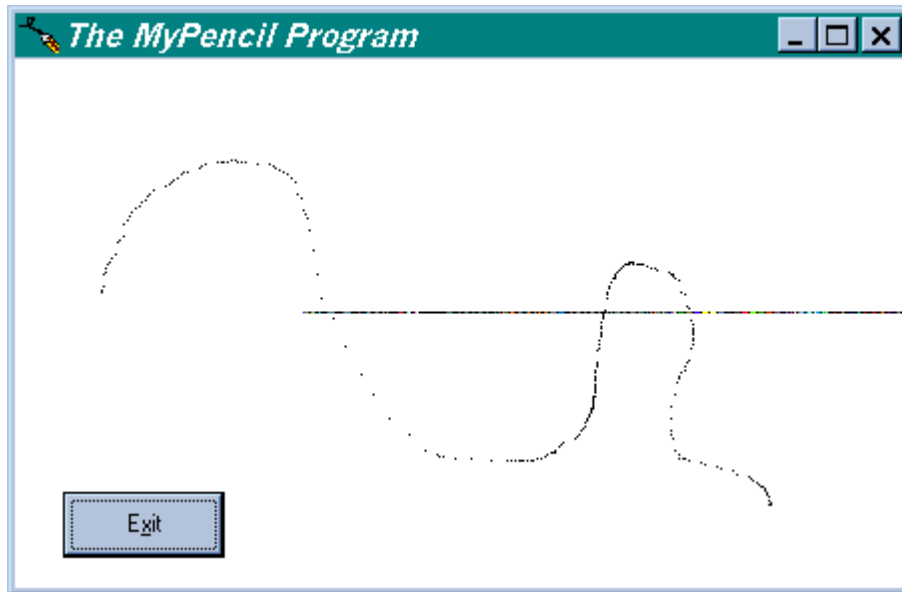


Figure 23.2. *The MyPencil program draws points according to the mouse movements.*

The thing to note about Figure 23.2 is that some portions of the path are dense (a lot of points next to each other), while other portions of the path are not dense. Why is that? A point is drawn only after the PC checked the mouse status. So as you move the mouse, a few point along the path of the movement were not drawn because the PC did not check the mouse status while the mouse was moved.

Because the PC checks the mouse status periodically at a regular time intervals, when the user moves the mouse fast, the PC misses a lot of points along the path of the mouse. This create a non-dense path of points.

Connecting the Dots

The path of points shown in Figure 23.2 is drawn according to the user mouse movement. As stated, if the user moves the mouse slowly, the points are drawn next to each other. However, it is unreasonable to assume that the user will move the mouse slowly.

Because it is unreasonable to assume that the user will draw by moving the mouse slowly, most drawing programs (such as Paint and Paintbrush) are connecting the dots with straight lines. You'll will now write code that draws straight lines according to the mouse movements.

=====

 Modify the code inside the **Form_MouseMove()** procedure so that it will look as follows:

```
Private Sub Form_MouseMove(Button As Integer, _
                            Shift As Integer, _
                            X As Single, _
                            Y As Single)

    If Button = 1 Then

        Form1.Line (Form1.CurrentX, Form1.CurrentY)-(X, Y)

    End If

End Sub
```

The code that now appears under the **If** statement draws a line:

```
Form1.Line (Form1.CurrentX, Form1.CurrentY)-(X, Y)
```

The line is drawn from the point that has the coordinates:

```
(Form1.CurrentX, Form1.CurrentY)
```

To the point that has the coordinates:

```
(X,Y)
```

Form1 has the properties **CurrentX** and **CurrentY**. These properties are examples of properties that are available only during runtime. You will not see these properties listed among the properties of Form1 inside the Properties window of Form1.

=====

Basically, **CurrentX** and **CurrentY** are used for various drawing methods. These coordinates hold the coordinates of the point that was last drawn on Form1.

Recall that **X** and **Y** are parameters of the **Form_MouseMove()** procedure. These coordinates are the coordinates of the mouse cursor at the time the **Form_MouseMove()** was executed.

So putting it altogether, when the user moves the mouse (while holding down the left button of the mouse), a line is drawn. The line is drawn from the last point that was drawn to the current location of the mouse

Let's see your code in action:

 Select **Save Project** from the **File** menu of Visual Basic to save your work.

 Execute the MyPencil program.


 Place the mouse cursor inside the window of the MyPencil program, hold down the left button of the mouse, and draw something.

Figure 23.3 shows an example of drawing accomplished with the MyPencil program.

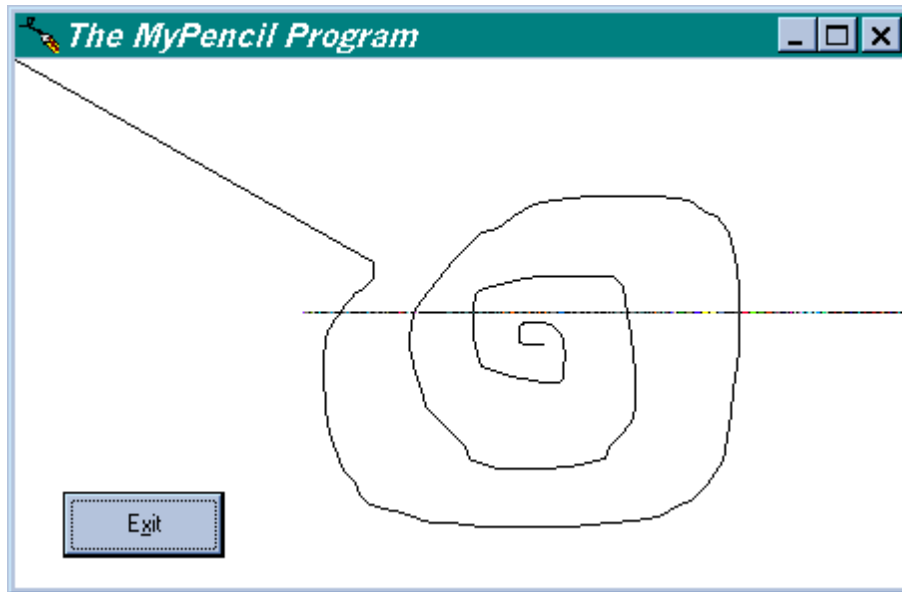



Figure 23.3. Drawing lines with the MyPencil program.


 Experiment with the MyPencil program, and then click its Exit button to terminate the program.

Fixing the MyPencil Program

Note that the first line in Figure 23.3 is drawn as a line that starts at coordinates (0,0) which is the upper left corner of the window. Why? Because the line that is drawn inside **Form_MouseMove()** starts at coordinates:

(Form1.CurrentX, Form1.CurrentY)

As stated, (Form1.CurrentX , Form1.CurrentY) are the coordinates of the last point that was drawn inside Form1. When **Form_MouseMove()** is executed for the first time, **CurrentX** and **CurrentY** are both **0** (because no point was drawn yet inside Form1). This is the reason the first line is a line that starts at (0,0). However, this can be easily fixed as follows:

 Type the following code inside the **Form_MouseDown()** procedure:

```
Private Sub Form_MouseDown(Button As Integer, _
=====
```

```
Shift As Integer, _  
X As Single, _  
Y As Single)
```

```
If Button = 1 Then  
    Form1.CurrentX = X  
    Form1.CurrentY = Y  
End If
```

End Sub

So the moment the user pressed down the left button of the mouse, the **CurrentX** and **CurrentY** properties are updated with the coordinates of the mouse cursor. Then the **Form_MouseMove()** procedure is executed, and a line is drawn from the point where the mouse cursor was located when the user pressed the left button of the mouse, to the point where the mouse cursor was located when the **Form_MouseMove()** procedure was executed.

Let's see your code in action:

 Select **Save Project** from the **File** menu of Visual Basic to save your work.

 Execute the MyPencil program.



 Place the mouse cursor inside the window of the MyPencil program, press the left button of the mouse down, and move the mouse.

Figure 23.4 is an example of a drawing accomplished with the MyPencil program

 Experiment with the MyPencil program, and then click the Exit button to terminate the program.

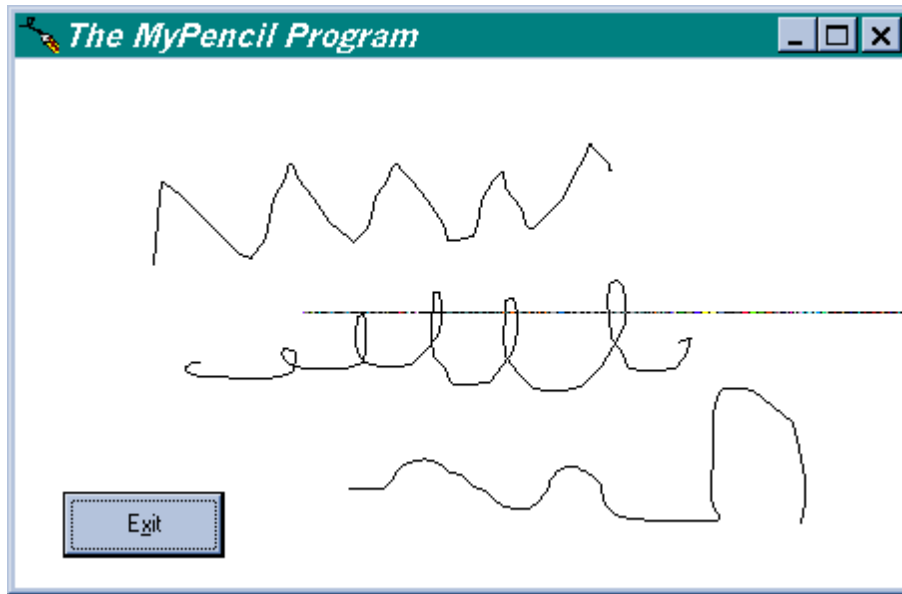




Figure 23.4. Drawing with the MyPencil program.

Changing the Mouse Cursor

Here is a little enhancement to the MyPencil program that you may or may not find attractive:

 At design time, set the **MouseIcon** property of Form1 to the **Pen01.ICO** file that resides inside the **\Icons\Writing** sub-directory of Visual Basic.

 At design time set the **MousePointer** property of Form1 to: **99-Custom**

 Execute the MyPencil program, and notice that the mouse cursor appears as a picture of the **Pen01.ICO** icon that you assigned to the **MouseIcon** property of Form1.

The reason we stated that you may or may not want to include such feature, is that a feature like the may confuse your user. Also, the line that is drawn is drawn from the center of the icon that serves as the mouse

cursor. So if you decide to include such a feature, make the picture of the icon so that the tip of the pen will be at the center of the icon picture.

Another reason why you may not like this feature, is that as you move the mouse, the icon picture that serves as the mouse cursor picture blinks (and this may annoy your user).

Using Cursor (CUR) files

In the preceding steps, you set the **MouseIcon** property of Form1 to an icon picture (Pen03.ICO) . Alternatively, you can assign a Cur file to the **MouseIcon** property. A Cur file is a file that contains a small picture. For example. MyCursor.Cur is an example of CUR file. Unlike an ICO file, the Cur file is a little smaller. Cur files are used as pictures for mouse cursor. You will not notice blinking while using Cur files.

What You Accomplished in This Lesson



You completed Lesson 23 of the Self Study Visual Basic tutorial. In this lesson you learned to write programs that let the user draw like the pencil tool of Paint (or Paintbrush).

Frequently Asked Questions



Q1. How can I make the MyPencil draw with a thicker pen?

A1. The **Line** method that you use to draw the lines "works" on Form1 as follows:


```
Form1.Line (Form1.CurrentX, Form1.CurrentY)-(X, Y)
```


=====

You can set a larger value for the **DrawWidth** property of Form1, and as a result, the line will be drawn with a thicker pen.

 Set the **DrawWidth** property of Form1 to 10.

 Select **Save Project** from the **File** menu of Visual Basic to save your work.

 Execute the MyPencil program, and note that a thicker pen is used (see Figure 23.5).

 Experiment with the MyPencil program, then click its Exit button to terminate the program.

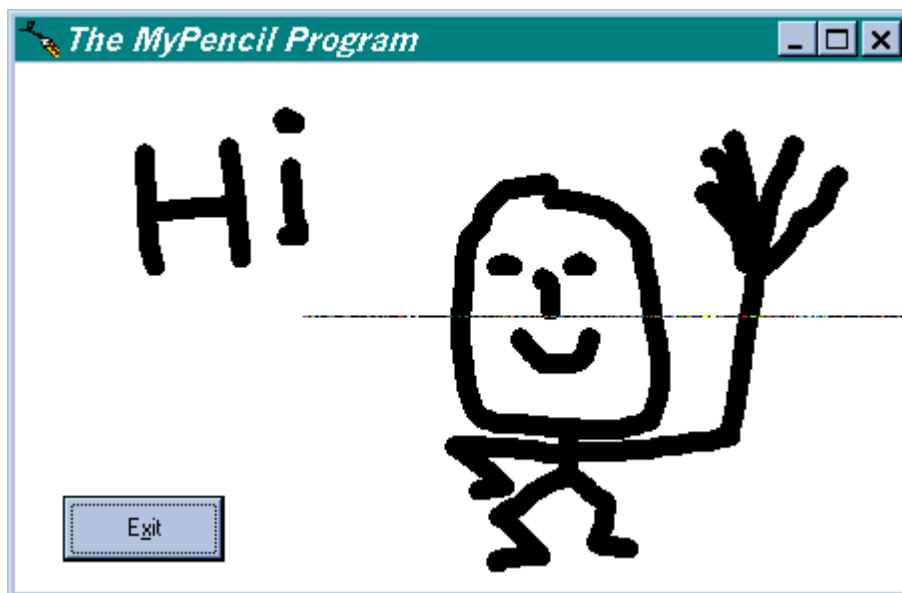


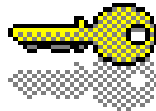
Figure 23.5. *The MyPencil program draws with a thicker pen.*

Exam



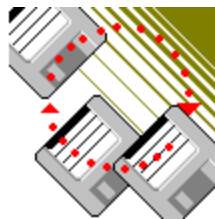
1. The **CurrentX** and **CurrentY** properties can be set during design time by setting these properties inside the Properties window of Form1:
 - (a) True
 - (b) False

Answers to Exam



1. (b) False. The **CurrentX** and **CurrentY** properties are examples of properties that can only be set during runtime.

Project



Modify the MyPencil program so that the red pen is used for the drawing.

 Modify the code inside the **Form_MouseMove()** procedure so that it will look as follows:

```
Private Sub Form_MouseMove(Button As Integer, _
                            Shift As Integer, _
                            X As Single, _
                            Y As Single)

If Button = 1 Then

Form1.Line (Form1.CurrentX, Form1.CurrentY)-(X, Y), _
           RGB(255, 0, 0)

End If

End Sub
```

You added the color parameter to the **Line** method as follows:


```
Form1.Line (Form1.CurrentX, Form1.CurrentY)-(X, Y), _
           RGB(255, 0, 0)
```


So now the **Line** method will draw using a red pen.

Let's see this in action:

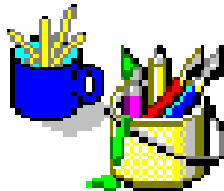
 Select **Save Project** from the **File** menu of Visual Basic to save your work.

 Execute the MyPencil program.

 Draw something with the MyPencil program, and note that red pen is used for the drawing.

 Experiment with the MyPencil program and then click its Exit button to terminate the program.

Cosmetic Considerations



● When designing program such as the MyPencil program, the client area (the available area inside the window of the program) should be dedicated to the drawing. Thus, do not place buttons such as the Exit button inside the window of the program. So where will you place the buttons of the program? You can design a menu to the program. Additionally, you can design a toolbar or status bar, and place your buttons inside the toolbar/status bar.

How To Contact TegoSoft



You can contact TegoSoft Inc. by any one of the following methods:

- Use TegoSoft Internet Web site:
<http://www.tegosoft.com>
- Send TegoSoft an e-mail:
tegosoft@msn.com
- Send TegoSoft a letter:


*TegoSoft Inc.
P.O.Box 389
Bellmore, NY 11710
USA*

Technical Support



If you have a technical question, you can post the question to the TegoSoft Technical Support staff, and they will try to answer your question.

The **best** way to post a technical question is by sending TegoSoft an e-mail.

 The e-mail of TegoSoft is:
tegosoft@msn.com

When sending TegoSoft an e-mail with a technical question, please follow the following format:

Date: _____
Your name: _____
Company (if applicable): _____
Your phone number: _____
Your e-mail: _____
Country (if not USA): _____
State (if inside USA): _____

Operating System used: _____
Programming language and version : _____

=====

My technical question is:

Copyright © and Notices

Copyright © 1994, 1995, 1996 by TegoSoft Inc. ® All Rights Reserved

TegoSoft Self Study Tutorials & Software

Copyright ©1994, 1995, 1996 by TegoSoft Inc. ® All Rights Reserved

Although precaution was taken in the preparation of this document, TegoSoft assumes no responsibility for errors or omissions. TegoSoft is not liable for damages resulting from the use of the information contained in this document. Use this document at your own risk. This document is copyright protected. This means that you should treat this document like any other copyright material. No part of this document should be copied in any way. You are not allowed to use this document or any part of this document for any purpose other than read it. You are not allowed to publish this document or any part of this document in any way. You are not allowed to sell this document or any part of this document, you are not allowed to incorporate this document or any part of this document in any book, magazine, Web Site, Electronic forums, disks, CDs, or any other media. The only thing that you are allowed to do with this document is read it for the sole purpose of studying the material that is presented in this document. If this document includes software, the software must be treated in the exact same way that this document is treated. You are not allowed to distribute the software in any way. The sole purpose of supplying the accompanying software is to enable you to experience with it. No part of this document should be modified. If accompanying software is included with this document, you are not allowed to modify the software.

Rev. 031796-1

=====