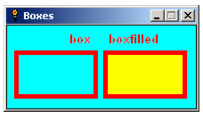
Easy Polygon via API

-
Alyce

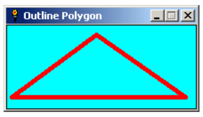
What is a Polygon?

A polygon is a closed plane figure bounded by straight sides. A polygon must have at least three sides. Two lines cannot create a closed figure. The sides need not be equal in length, and there can be many, many sides. Liberty BASIC has native graphics commands to draw some specialized polygons. We can draw four-sided polygons whose angles are right angles. We do this with the "box" and "boxfilled" commands. If all sides are equal, the figure is a square. The sides of the box are always parallel to the sides of the window. In other words, we cannot set a box on the diagonal and create a diamond. The "box" command draws the outline of a box. It is drawn in the current "color" and the lines are the current"size." The "boxfilled" command draws a filled box. The outline is created in the same way as for the "box", but the inside is filled with the current "backcolor."



Drawing Polygon Outlines

We can draw a polygon in any orientation, with any number of sides, if we use the "line" command, or if we use turtle graphics. Here is an example of the simplest polygon: the triangle.



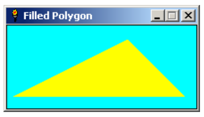
```
nomainwin
WindowWidth=220:WindowHeight=120
open "Outline Polygon" for graphics_nsb_nf as #1
#1 "trapclose [quit]"
#1 "down; fill cyan"
#1 color red; size 5"
#1 "backcolor yellow"
#1 "line 100 10 200 80"
#1 "line 200 80 5 80"
```

```
#1 "line 5 80 100 10"
wait

[quit] close #1:end
```

Drawing Filled Polygons

Although we issued the command "backcolor yellow", the polygon is not filled with yellow. We can create a filled polygon by drawing it in a loop, decreasing the size each time through the loop. The code gets complicated really quickly, though, and large figures would take a while to complete. Alternately, we can draw an outline, and use the FloodFill API call to fill it, but there is a much better way to draw filled polygons by API, which is detailed later in this topic.



```
nomainwin
WindowWidth=220:WindowHeight=120
open "Filled Polygon" for graphics_nsb_nf as #1
#1 "trapclose [quit]"
#1 "down; fill cyan"

x2=200:x1=5
y=80

#1 "color yellow; size 1"

while x1 < x2
    y=y-1
    x1=x1+2
    x2=x2-1
    #1 "line ";x1;" ";y;" ";x2;" ";y
wend

    #1 "flush"
wait

[quit] close #1:end
```

API Outline Polygons

All figures drawn with API calls to GDI (Graphics Device Interface) are filled! The only way to draw an outline polygon with API calls is to create a special transparent brush with which to fill the polygon. It must be created, selected into the device context, then later de-selected from the device context and destroyed. It is easier to draw outline polygons with native Liberty BASIC commands, like "line" or "go" or "goto".

Drawing Filled Polygons the EASY Way

Filled polygons are most easily drawn with API calls.

The Device Context

When we issue API graphics commands, we do not send them to a window. We send them to GDI by way of a construction known as a Device Context. This is the "interface" in Graphics Device Interface (GDI). GDI acts as an interface between our code and the hardware. We don't need to know the specifics of the hardware. How many colors are supported by the display screen? We don't need to know! We let GDI make any needed color adjustments for us. To draw with GDI, we first issue a call to GetDC, which takes the handle of the graphics window or graphicbox as an argument, and returns the handle of its Device Context (DC). We then issue our drawing commands to GDI through this Device Context Handle, commonly referred to as hDC. When an hDC is no longer needed, or when the program ends, a call must be made to ReleaseDC.

Pens and Brushes

GDI allows you to create several kinds of pens with which to draw figures. It isn't necessary to use GDI to do this, though. The pen will be the color we set when we issue a "color" graphics command. It will be the size we set with the "size" graphics command. GDI also allows us to create several kinds of brushes with which to fill drawn figures. Again, it isn't necessary to create brushes. The figures will be filled with the color set when we issue a "backcolor" command.

The PolyPoints STRUCT

The key to drawing a polygon with GDI is the creation of a STRUCT that contains the pairs of points. It looks like this:

```
STRUCT PolyPoints,_
```

```
    x1 as long,_
    y1 as long,_
    x2 as long,_
    y2 as long,_
    x3 as long,_
    y3 as long
```

Notice that the members of the struct come in x,y pairs. There can be as many pairs of points as needed to draw the polygon desired. For demonstration purposes in this article, we're keeping it short and simple by using the minimum, which is three pairs of points, or six members in the struct. The Polygon API call needs to know how many vertices are contained in the polygon. That number is equal to half the number of members in the struct. Remember, each vertex consists of an xy pair. For a triangle we need six struct members to create THREE vertices:

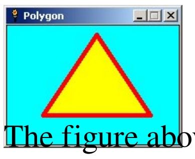
```
nCount=3   'number of x,y pairs in STRUCT
```

The xy coordinates of the vertices of the polygon are set when the struct is filled.

```
    PolyPoints.x1.struct = 100
    PolyPoints.y1.struct = 10
    PolyPoints.x2.struct = 160
    PolyPoints.y2.struct = 100
    PolyPoints.x3.struct = 40
    PolyPoints.y3.struct = 100
```

The members can be filled with literal numbers or with variables.

The Polygon API Function



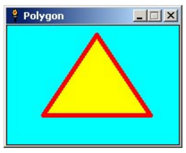
The figure above is created with this simple API call:

```
calldll #gdi32, "Polygon",_
hdc as ulong,_         'device context of window or control
PolyPoints as struct,_'array of points
nCount as long,_      'number of x,y pairs in array
```

```
result as long
```

Demo One

Below, you will find a simple demo program that draws a triangle. It looks like this:



```
nomainwin
WindowWidth=200:WindowHeight=160
open "Polygon" for graphics_nsb_nf as #1
#1 "trapclose [quit]"
#1 "down; fill cyan"
#1 "color red; size 5"  'will be used to outline figure
#1 "backcolor yellow"   'will be used to fill figure

h=hwnd(#1)  'window handle

'get device context for window:
calldll #user32, "GetDC",_
h as ulong,_ 'window handle
hdc as ulong 'returns handle to device context

STRUCT PolyPoints,_
    x1 as long,_
    y1 as long,_
    x2 as long,_
    y2 as long,_
    x3 as long,_
    y3 as long

nCount=3   'number of x,y pairs in STRUCT

'The STRUCT must be filled before it can be used in an api call:

    PolyPoints.x1.struct = 100
    PolyPoints.y1.struct = 10
    PolyPoints.x2.struct = 160
    PolyPoints.y2.struct = 100
    PolyPoints.x3.struct = 40
    PolyPoints.y3.struct = 100
```

```
calldll #gdi32, "Polygon",_
hdc as ulong,_        'device context of window or control
PolyPoints as struct,_'array of points
nCount as long,_      'number of x,y pairs in array
result as long

calldll #user32, "ReleaseDC",_
h as ulong,_    'window handle
hdc as ulong,_  'device context
ret as long

'method to flush GDI graphics:
#1 "getbmp pix 0 0 180 120"
#1 "drawbmp pix 0 0;flush"

wait

[quit] close #1:end
```

Demo Two

Here an amusing demo that allows the user to click three times with the mouse to set the vertices of a triangle. After the third mouse click, the triangle is drawn. This program demonstrates the way to fill the struct members with variables.

```
nomainwin
WindowWidth=400:WindowHeight=400
open "Draw a Triangle" for graphics_nsb_nf as #1
#1 "trapclose [quit]"
#1 "down; fill cyan"
#1 "color red; size 5"  'will be used to outline figure
#1 "backcolor yellow"   'will be used to fill figure
#1 "setfocus;when leftButtonDown [draw]"
#1 "place 10 30"
#1 "\Click three spots to draw a triangle."

h=hwnd(#1)  'window handle

'get device context for window:
calldll #user32, "GetDC",_
```

```
h as ulong,_ 'window handle
hdc as ulong 'returns handle to device context

STRUCT PolyPoints,_
    x1 as long,_
    y1 as long,_
    x2 as long,_
    y2 as long,_
    x3 as long,_
    y3 as long
nCount=3   'number of x,y pairs in STRUCT

wait

[draw]
    count=count+1
    if count=1 then
        'remove text
        #1 "fill cyan"
        'draw vertex
        #1 "set ";MouseX;" ";MouseY
        'fill first xy pair in struct with mouse coords
        PolyPoints.x1.struct=MouseX
        PolyPoints.y1.struct=MouseY
        wait
    end if

    if count=2 then
        'draw vertex
        #1 "set ";MouseX;" ";MouseY
        'fill second xy pair in struct with mouse coords
        PolyPoints.x2.struct=MouseX
        PolyPoints.y2.struct=MouseY
        wait
    end if

    if count=3 then
        'draw vertex
        #1 "set ";MouseX;" ";MouseY
        'fill struct with last point xy
        PolyPoints.x3.struct=MouseX
        PolyPoints.y3.struct=MouseY
        count=0 'reset counter variable
    end if

    'after the third mouse click, draw the triangle
    calldll #gdi32, "Polygon",_
    hdc as ulong,_        'device context of window or control
    PolyPoints as struct,_'array of points
    nCount as long,_      'number of x,y pairs in array
    result as long

    'method to flush GDI graphics:
    #1 "place 10 30"
    #1 "\Click three spots to draw a triangle."
    #1 "segment segID"
    #1 "delsegment segID"
    #1 "getbmp pix 0 0 400 400"
    #1 "drawbmp pix 0 0;flush"

    wait

[quit]
close #1

calldll #user32, "ReleaseDC",_
h as ulong,_     'window handle
hdc as ulong,_   'device context
ret as long

end
```

Demo 3 and Demo 4

Stefan Pendl has shared two enchanced versions of the polygon code. The methods are for programmers with advanced API skills. The methods make it possible to easily draw polygons with many points. Find them here:

|Stefan's Advanced Polygon API Demos

Easy Polygon via API | What is a Polygon? | Drawing Polygon Outlines | Drawing Filled Polygons | API Outline Polygons | Drawing Filled Polygons the EASY Way | The Device Context | Pens and Brushes | The PolyPoints STRUCT | The Polygon API Function | Demo One | Demo Two | Demo 3 and Demo 4