C++ Graphics Programming Guide

Overview

The C++ Graphics Library (**graphLib1.lib**) and header file (**graph1.h**) are required before developing graphics applications using .Net 2008. Download and add these files to your .Net 2008 project as described on page 9 of the *VisualStudioTutorial* guide. This guide will describe the usage of all functions found in the C++ Graphics library.

Function Guide

The following is a complete list and description of all functions found in the C++ Graphics Library. The functions are grouped by category. Example usage of each function is provided on page 7.

Window Initialization Functions

```
void displayGraphics();
```
This function creates an empty 640x480 window with a solid black background. **This function must be invoked first before using any other graphics function.**

```
void clearGraphics();
```
This function removes all objects currently displayed in the graphics window. The original solid background is redisplayed upon completion of this function.

Text Display Object/Function

The `gout`(pronounced *g*ee *out* - abbreviation for *graphics output*) object is used for displaying textual data in the graphics window. The usage of this object is very similar to that of `cout` and is illustrated below:

```
gout << setPos(x_coord,y_coord) << literal/variable <<
literal/variable << …… << endg;
```

The *setPos* manipulator is used **only** with `gout` and specifies where the *following* data will be displayed within the graphics window (e.g., *x_coord*/*y_coord* are the coordinates where the text is displayed). *setPos* should be placed immediately *after* the first << delimiter and before any data variable or literal. Each variable or literal is delimited by the << operator identical to `cout` usage. The `endg` identifier is **required** at the end of the statement and *flushes* the internal graphics buffer thus ensuring that the data is properly displayed within the graphics window.

```
void clearText();
```
This function removes **all** previously displayed textual data within the graphics window.

Object Creation Functions

The following functions create graphic objects and display them at the specified coordinates within the graphics window. Each object is assigned a unique identifier that can be used when moving, coloring, replacing, or removing the object. This unique identifier is always returned by the object creation functions.

```
int drawPoint(int x, int y);
```
This function draws a *white* point in the graphics windows at the coordinate specified by the two parameters.

Input Parameters:
`int x` - x coordinate of the point.
`int y` - y coordinate of the point.

Return Value
Returns an integer representing the object number for the newly created point.

```
int drawLine(int x1, int y1, int x2, int y2, int width);
```
This function draws a line between the 2 specified points (x1,y1) and (x2,y2).

Parameters:
`int x1` - x coordinate of the first point of the line
`int y1` - y coordinate of the first point of the line
`int x2` - x coordinate of the second point of the line
`int y2` - y coordinate of the second point of the line
`int width` - width of a the specified line *(greater than 1 results in a rectangle)*

Return Value
Returns an integer representing the object number for the newly created line.

```
int drawRect(int x1, int y1, int width, int height);
```
This function draws a rectangle *parallel to the horizontal and vertical axis.*.

Parameters:
`int x1` - upper left x coordinate of the rectangle
`int y1` - upper left y coordinate of the rectangle
`int width` - horizontal width of the rectangle
`int height` - vertical height of the rectangle

Return Value
Returns an integer representing the object number for the newly created rectangle.

```
int drawCircle(int radius, int x, int y);
```
This function draws a circle that has a specified radius and center provided by a point (the point is specified by its x/y coordinates).

Parameters:
`int radius` - radius of the circle
`int x` - x coordinate of the circle's center point
`int y` - y coordinate of the circle's center point

Return Value
Returns an integer representing the object number for the newly created circle

---

```
int displayBMP(char* fn,int x, int y);
```
This function draws a 24-bit Windows Bitmap Image at the specified x/y coordinates.

Parameters:
`char* fn` - filename of the bitmap image. (e.g. "image1.bmp")
`int x` - upper left x coordinate where the image is to be drawn
`int y` - upper left y coordinate where the image is to be drawn

Return Value
Returns an integer representing the object number for the newly created bitmap image

---

Object Modification Functions
The following functions modify graphic objects previously created. Each of the modification functions requires the unique object number returned from one of the object creation functions described in the previous section.

```
void setColor(int obj_no, int r, int g, int b);
```
This function sets the color of a previously created graphic object identified by `obj_no`. The color is based on the RGB color space

Parameters:
`int obj_no` - The unique object number corresponding to a graphic object
`int r` - The red component of the RGB color (valid values between 0-255)
`int g` - The green component of the RGB color (valid values between 0-255)
`int b` - The blue component of the RGB color (valid values between 0-255)

```
void moveObject(int obj_no, int x, int y);
```
This function moves a previously created graphic object to a specified point defined by its x/y coordinates.

Parameters:
`int obj_no` - The unique object number corresponding to a graphic object
`int x` - The location of the x coordinate where the object will move to.
`int y` - The location of the y coordinate where the object will move to.

```
void removeObject(int obj_no);
```
This function **permanently** removes an object from the graphic window. *All memory associated with the specified object is freed.*

Parameters:
`int obj_no` - The unique object number corresponding to a graphic object

---

```
void replaceObject(int orig_obj_no, int new_obj_no);
```
This function **temporarily** replaces one existing object with another one. No memory is released for either specified object.

Parameters:
`int orig_obj_no` - The unique object number corresponding to the currently displayed graphic object

`int new_obj_no` - The unique object number corresponding to the new object to be displayed

---

Event Functions
The following functions are used for processing events generated from the keyboard or the mouse.

```
bool up();
```
This function tests if the **up** arrow key was pressed.

Return Value
Returns *true* if **up** arrow key is pressed, *false* otherwise.

```
bool down();
```
This function tests if the **down** arrow key was pressed

Return Value

Returns *true* if **down** arrow key is pressed, *false* otherwise.

```
bool left();
```
This function tests if the **left** arrow key was pressed.

Return Value
Returns *true* if **left** arrow key is pressed, *false* otherwise.

```
bool right();
```
This function tests if the **right** arrow key was pressed.

Return Values
Returns *true* if **right** arrow key is pressed, *false* otherwise.

```
bool leftMouse(int&x, int&y);
```
This function tests if **left mouse button** was clicked *once* (quickly pressed and released)

Return Values
Returns *true* if **left mouse button** clicked, *false* otherwise.
`int&x` - set to the x coordinate on the graphics screen where the mouse was clicked
`int&y` - set to the y coordinate on the graphics screen where the mouse was clicked

---

```
bool rightMouse(int&x, int&y);
```
This function tests if **right mouse button** was clicked once (quickly pressed and released)

Return Values
Returns *true* if **right mouse button** clicked, *false* otherwise.
`int&x` - set to the x coordinate on the graphics screen where the mouse was clicked
`int&y` - set to the y coordinate on the graphics screen where the mouse was clicked

---

```
bool middleMouse(int&x, int&y);
```
This function tests if **middle mouse button** was clicked once (quickly pressed and released)

Return Values
Returns *true* if **middle mouse button** clicked, *false* otherwise.

`int&x` - set to the x coordinate on the graphics screen where the mouse was clicked

`int&y` - set to the y coordinate on the graphics screen where the mouse was clicked

---

```
bool mouseDragged(int& x, int& y);
```

This function tests if the **left mouse button** is pressed down while the mouse is moving (i.e., *dragged*) across the graphics screen.

Return Values

Returns *true* if mouse is dragged, *false* otherwise.

`int&x` - set to the x coordinate on the graphics screen where the mouse was dragged (x is continually changed as the mouse is dragged)

`int&y` - set to the y coordinate on the graphics screen where the mouse was dragged (y is continually changed as the mouse is dragged)

Miscellaneous Functions

```
void getPos(int obj_no, Point points[], int& no_points);
```

This function returns all relevant points associated with a specified object identified by `obj_no`.

Input Parameters

`int obj_no` - The unique object number corresponding to a graphic object

Return Values

`Point points[]` - Array of structures declared with the `Point` structure. This array contains all relevant *points* used for specifying the location of the object.

`int& no_points` - set to the number of elements contained in the *points* array variable.

```
void GRAPH_SS();
```

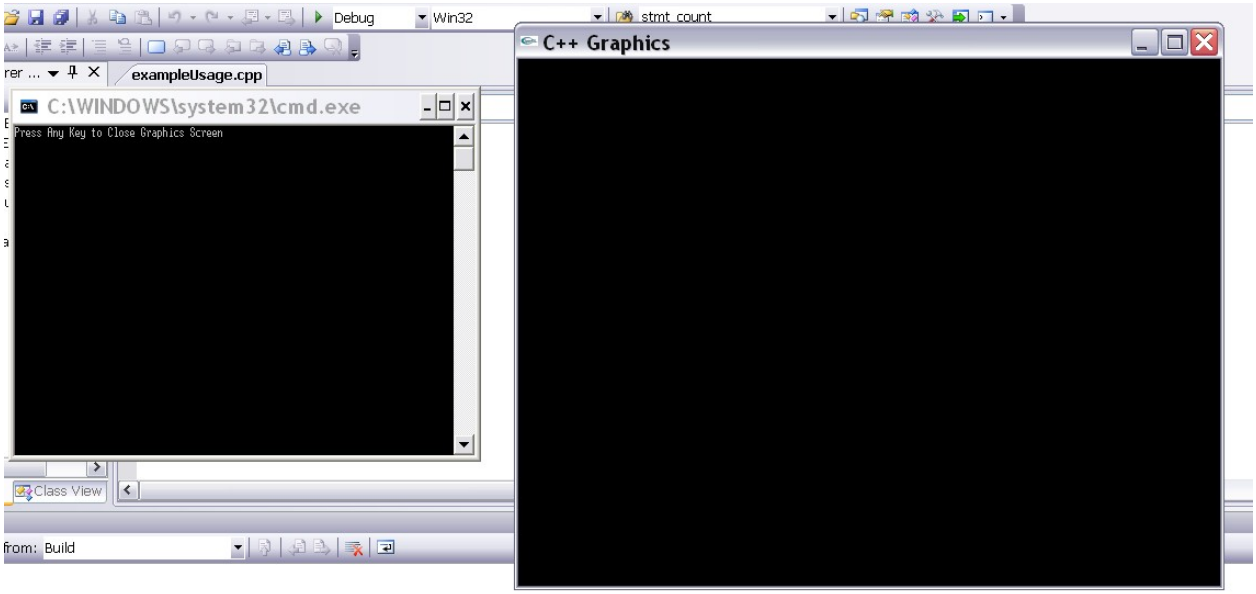
This function is useful when Single Stepping graphic functions within the Microsoft .Net 2008 Debugger. Calling this function immediately after a graphic's statement synchronizes the debugger with the Graphics Window thus allowing the results of the graphic's statement to be viewed instantly.

Example Usage of Graphic Functions

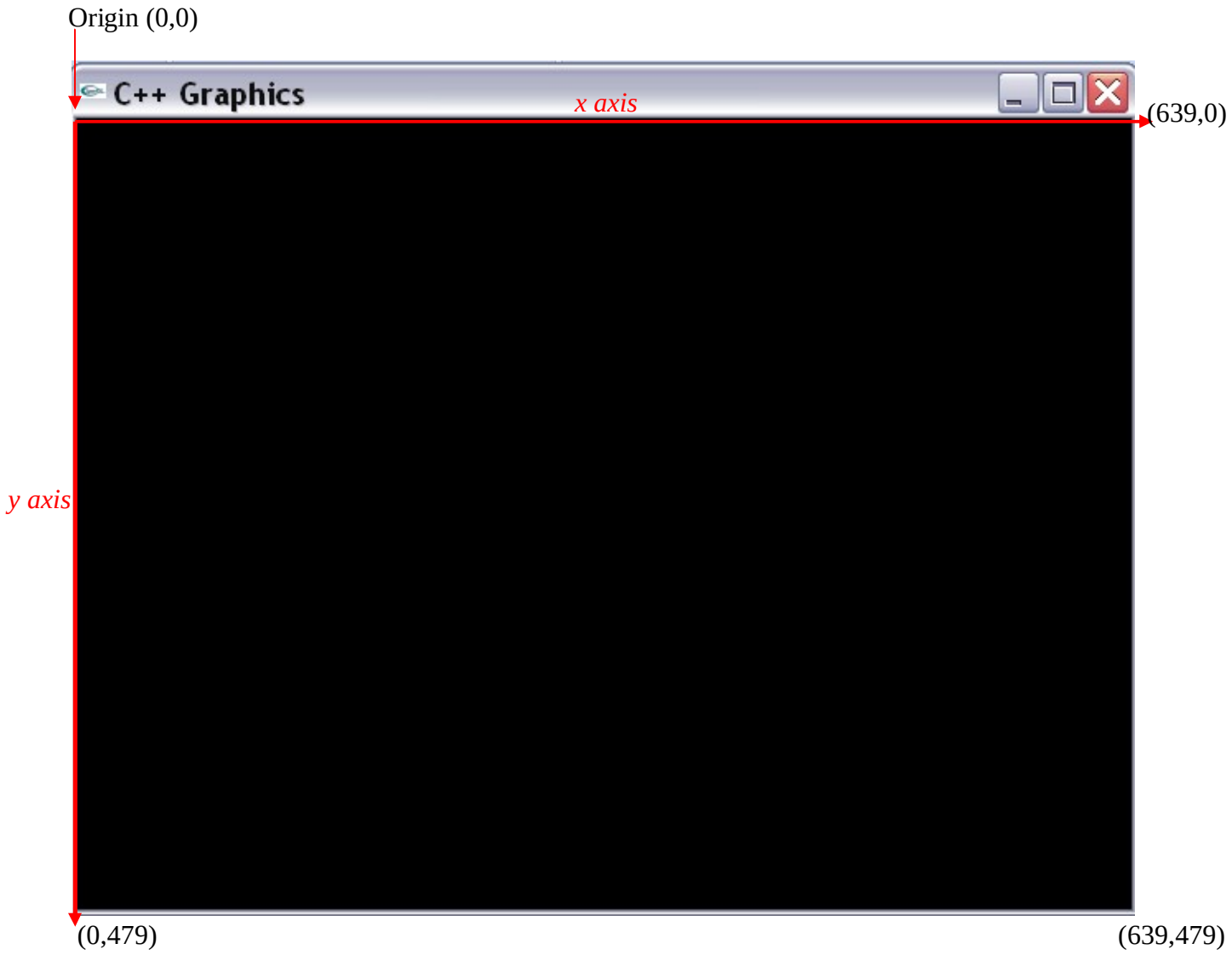
The most basic program that displays only the empty graphics window is shown below:

```
#include <iostream>
#include "graph1.h"

using namespace std;

int main()
{
  //Display empty graphics window
  displayGraphics();

  return 0;
}
```

Two windows (console/graphic) are displayed when running the above program:



The coordinate system and dimensions of the C++ Graphics window on the right is shown below in more detail:



Press enter at any time to close all windows.

Creating Objects

We next want to write a program that will display the following:

A circle with radius of 25 pixels located at position (75,75)
A rectangle located at position (300,300) with a width of 25, and a height of 50
A blue line from the origin (0,0) to the midpoint of the screen (320,240)
Text placed beneath the rectangle displaying the area of the rectangle
An image of a blue ball located at position (400,400).

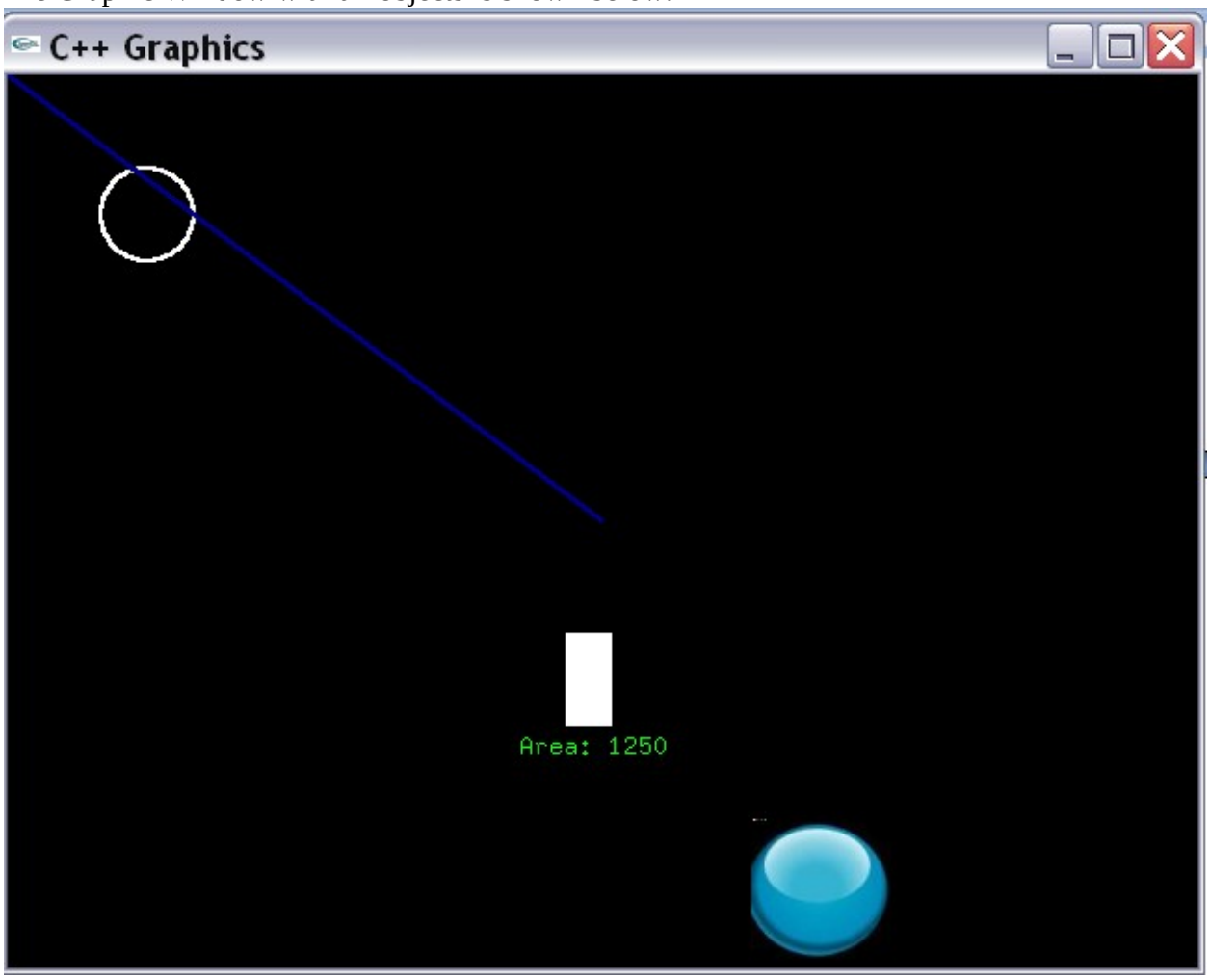
The image of blue ball is shown below:

The program that performs this is shown below:

```
#include <iostream>
#include "graph1.h"

using namespace std;

int main()
{
  //Variable Declaration/Initialization
  int obj_no = 0;
  int width = 0;
  int height = 0;

  //Display empty graphics window
  displayGraphics();

  //Prompt for the width/height of the rectangle
  cout << "Enter width/height of the rectangle:";
  cin >> width >> height;

  //Display a circle (could also prompt for the radius and coordinates!)
  drawCircle(25,75,75);

  //Display a Rectangle
  drawRect(300,300,25,50);

  //Display text pertaining to the area of the rectangle (
  gout << setPos(275,365) << "Area: " << width*height << endg;

  //Display a Line
  obj_no = drawLine(0,0,320,240,1);

  //Set the color of the line to mid-bright blue
  setColor(obj_no,0,0,128);

  //Display the image
  displayBMP("ball.bmp",400,400);


  return 0;
}
```

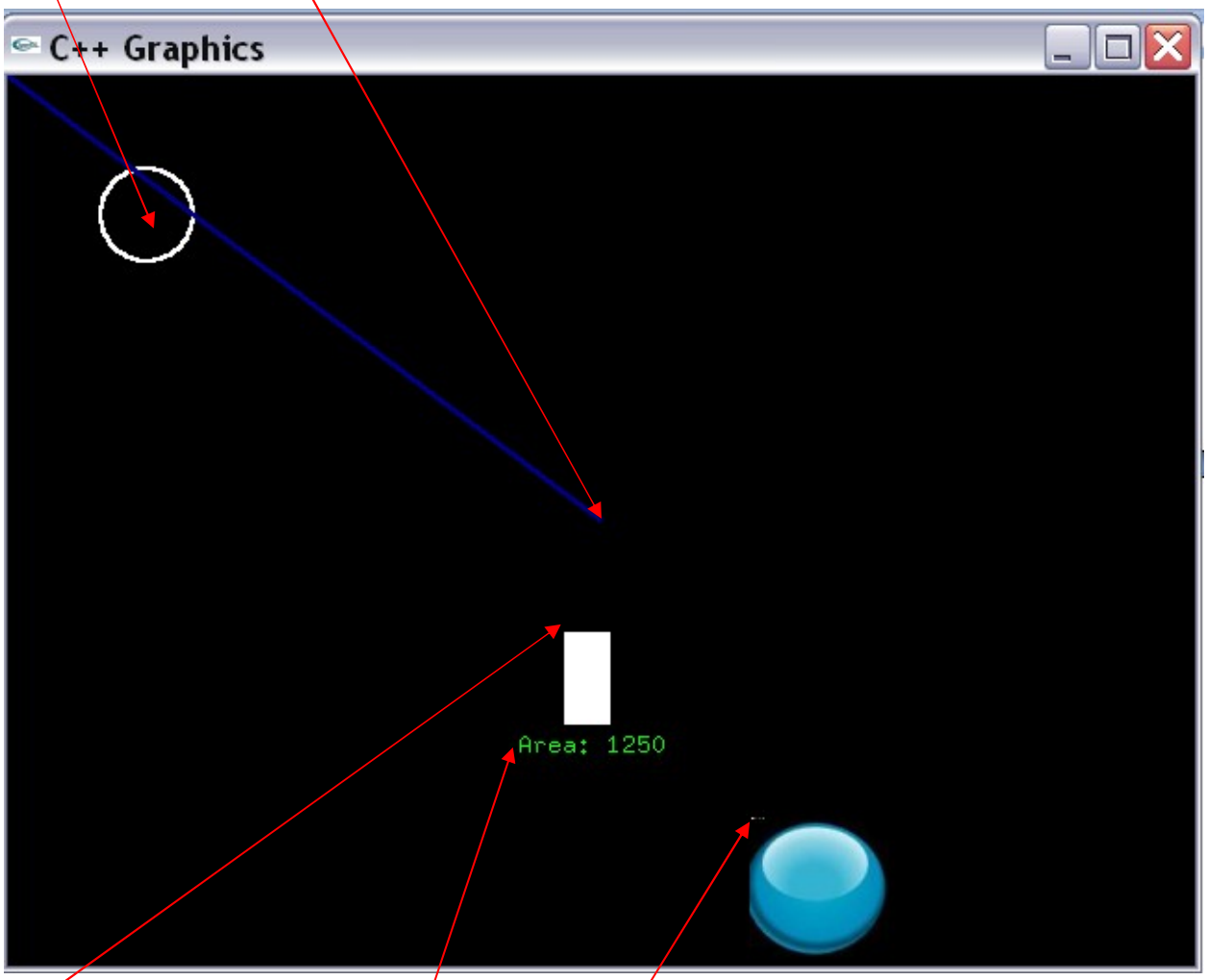
Note: You can mix cin/cout statements for input/output with any graphics library statement

The Graphic Window with all objects is shown below:



The Graphics Window is redisplayed with coordinates marked

(75,75) (320,240)



(300,300)
(upper left corner)

(275,365)
(Lower left of
the “A” in “Area”)

(400,400)

Moving Objects

Modify the previous program so the blue ball moves to the top of the screen and then back down.

The program that does this is shown below (modified lines highlighted in yellow):

```
#include <iostream>
#include "graph1.h"

using namespace std;

int main()
{
  //Variable Declaration/Initialization
  int i = 0;
  int obj_no = 0;
  int ball_obj = 0;
  int width = 0;
  int height = 0;

  //Display empty graphics window
  displayGraphics();

  //Prompt for the width/height of the rectangle
  cout << "Enter width/height of the rectangle:";
  cin >> width >> height;

  //Display a circle (could also prompt for the radius and coordinates!)
  drawCircle(25,75,75);

  //Display a Rectangle
  drawRect(300,300,25,50);

  //Display text pertaining to the area of the rectangle
  gout << setPos(275,365) << "Area: " << width*height << endg;

  //Display a Line
  obj_no = drawLine(0,0,320,240,1);

  //Set the color of the line to mid-bright blue
  setColor(obj_no,0,0,128);

  //Display the image
  ball_obj = displayBMP("ball.bmp",400,400);

  //Move the ball to the top
  for (i = 400; i >= 0; i--)
  {
    moveObject(ball_obj,400,i);
  }

  //Move the ball back down
  for(i = 0; i < 400; i++)
  {
    moveObject(ball_obj,400,i);
  }
```

```
 return 0;
}
```

Run the above program and observe how the ball moves from bottom to top and then back down.

Removing Objects

Modify the previous program so the ball is removed immediately once it returned to the bottom.

Just add the appropriate *removeObject* statement to the program (immediately before the return statement) as shown below:

```
  //Remove the ball
  removeObject(ball_obj);

  return 0;
```

(Removing an object also releases all memory associated with that object)

Events - Left Arrow

Modify the above program so the blue ball is moved to the left 5 pixels each time the **left arrow** is pressed. Pressing the **right arrow** will move the ball 5 pixels to the right. Pressing the **up arrow** key will allow the ball to move to the top as before. The complete program that performs this is shown below (added code highlighted in yellow):

```
#include <iostream>
#include "graph1.h"

using namespace std;

int main()
{
  //Variable Declaration/Initialization
  int i = 0;
  int obj_no = 0;
  int ball_obj = 0;
  int width = 0;
  int height = 0;
  int posx = 400;
  int posy = 400;


  //Display empty graphics window
  displayGraphics();

  //Prompt for the width/height of the rectangle
  cout << "Enter width/height of the rectangle:";
  cin >> width >> height;
```

```
  //Display a circle (could also prompt for the radius and coordinates!)
  drawCircle(25,75,75);

  //Display a Rectangle
  drawRect(300,300,25,50);

  //Display text pertaining to the area of the rectangle   gout <<
  setPos(275,365) << "Area: " << width*height << endg;

  //Display a Line
  obj_no = drawLine(0,0,320,240,1);

  //Set the color of the line to mid-bright blue
  setColor(obj_no,0,0,128);

  //Display the image
  ball_obj = displayBMP("ball.bmp",posx,posy);

  //Check for keyboard events
  do
  {
    //Check for left arrow
    if (left())
    {
      //Move to left 5 pixels
      posx -= 5;
      moveObject(ball_obj,posx,posy);
    }
    //Check for right arrow
    else if (right())
    {
      posx += 5;
      moveObject(ball_obj,posx,posy);
    }

  }while(!up()); //Check for up arrow to stop the loop!


  //Move the ball to the top
  for (i = posy; i >= 0; i--)
  {
    moveObject(ball_obj,posx,i);
  }

  //Move the ball back down
  for(i = 0; i < 400; i++)
  {
    moveObject(ball_obj,posx,i);
  }

  //Remove the ball
  removeObject(ball_obj);

  return 0;
}
```

NOTE: *You must have the Graphics Window **active** before any events will be recognized by the program!* (**Click** on the Graphics Window to make it active)

Events - Mouse Click

Modify the above program so that a *left* mouse click anywhere on the blue ball allows it to move to the top (instead of the up arrow as before). The complete program is shown below (modifications highlighted in yellow)

```
#include <iostream>
#include "graph1.h"

using namespace std;

int main()
{
  //Variable Declaration/Initialization
  const int BALL_WIDTH = 75;
  const int BALL_HEIGHT = 75;
  int i = 0;
  int x = 0;
  int y = 0;
  int obj_no = 0;
  int ball_obj = 0;
  int width = 0;
  int height = 0;
  int posx = 400;
  int posy = 400;
  bool exit_loop = false;


  //Display empty graphics window
  displayGraphics();

  //Prompt for the width/height of the rectangle
  cout << "Enter width/height of the rectangle:";
  cin >> width >> height;

  //Display a circle (could also prompt for the radius and coordinates!)
  drawCircle(25,75,75);

  //Display a Rectangle
  drawRect(300,300,25,50);

  //Display text pertaining to the area of the rectangle
  gout << setPos(275,365) << "Area: " << width*height << endg;

  //Display a Line
  obj_no = drawLine(0,0,320,240,1);

  //Set the color of the line to mid-bright blue
  setColor(obj_no,0,0,128);
```

```
  //Display the image
  ball_obj = displayBMP("ball.bmp",posx,posy);

  //Check for keyboard events
  do
  {
    if (left())
    {
      //Move to left 5 pixels
      posx -= 5;
      moveObject(ball_obj,posx,posy);
    }
    else if (right())
    {
      posx += 5;
      moveObject(ball_obj,posx,posy);
    }

   //Check for mouse click
   if (leftMouse(x,y))
   {

      //Make sure x/y coordinates inside of ball
      //(dimensions of ball is 75x75)
      if ( (x > posx) && (y > posy) && (x < (posx+ BALL_WIDTH)) && (y <
(BALL_HEIGHT+posy)) )
      {
        //break from loop
        exit_loop  = true;
      }
      else
      {
        exit_loop = false;
      }
    }

  }while(!exit_loop);


  //Move the ball to the top
  for (i = posy; i >= 0; i--)
  {
    moveObject(ball_obj,posx,i);
  }

  //Move the ball back down
  for(i = 0; i < 400; i++)
  {
    moveObject(ball_obj,posx,i);
  }

  //Remove the ball
  removeObject(ball_obj);

  return 0;
}
```

Dragging the Mouse

The previous program is modified so the blue ball can be *dragged* to any location on the screen by holding down the left mouse button over the blue ball and moving it (i.e., *dragging*) continuously. Pressing the left/right arrows move the ball as previously. Pressing the up arrow allows the ball to float to the top of the screen as before. The completed program is shown below:

```
#include <iostream>
#include "graph1.h"

using namespace std;

int main()
{
  //Variable Declaration/Initialization
  int i = 0;
  int x = 0;
  int y = 0;
  int obj_no = 0;
  int ball_obj = 0;
  int width = 0;
  int height = 0;
  int posx = 400;
  int posy = 300;
  int rect_obj = 0;
  bool mouse_dragged = false;
  bool exit_loop = false;


  //Display empty graphics window
  displayGraphics();

  //Prompt for the width/height of the rectangle
  cout << "Enter width/height of the rectangle:";
  cin >> width >> height;

  //Display a circle (could also prompt for the radius and coordinates!)
  drawCircle(25,75,75);

  //Display a Rectangle
  rect_obj = drawRect(300,300,25,50);

  //Display text pertaining to the area of the rectangle
  gout << setPos(275,365) << "Area: " << width*height << endg;

  //Display a Line
  obj_no = drawLine(0,0,320,240,1);

  //Set the color of the line to mid-bright blue
  setColor(obj_no,0,0,128);

  //Display the image
```

```
ball_obj = displayBMP("ball.bmp",posx,posy);

//Check for keyboard/mouse events
do
{
  if (left())
  {
    //Move to left 5 pixels
    posx -= 5;
    moveObject(ball_obj,posx,posy);
  }
  else if (right())
  {
    posx += 5;
    moveObject(ball_obj,posx,posy);
  }
  else if (up())
  {
    //Break out of loop
    exit_loop = true;
  }

  //Move the rectangle by dragging the mouse
  mouse_dragged = false;
  do
  {
    //Test if mouse dragged
    if (mouseDragged(x,y))
    {
      posx = x;
      posy = y;
      moveObject(ball_obj,posx,posy);
      mouse_dragged = true;
    }
    else
    {
      mouse_dragged = false;
    }

  }while(mouse_dragged);

}while(!exit_loop);


//Move the ball to the top
for (i = posy; i >= 0; i--)
{
  moveObject(ball_obj,posx,i);
}

//Move the ball back down
for(i = 0; i < 400; i++)
{
  moveObject(ball_obj,posx,i);
}

//Remove the ball
```

```
  removeObject(ball_obj);

  return 0;
}
```