iGraphics: A Wrapper For OpenGL in 2D

S. M. Shahriar Nirjon
University of Virginia
email: smn8z@virginia.edu
Date: August 8, 2008

Abstract

iGraphics.h header file contains some drawing functions that can be used to draw basic graphical shapes in Visual C++. These functions are implemented in openGL. Users of iGraphics do not need any knowledge of openGL to use it. Simply calling the drawing functions a user can draw any 2D shape on screen. This library also provides easy ways for animation, keyboard and mouse event handling.

1. Setup

Just copy the iGraphics folded in your PC. The folder contains the following files- GLUT.H, GLUT32.LIB, GLUT32.DLL, iGraphics.h, iMain.cpp and iDoc.pdf.

2. Description of iMain.cpp

Users of iGraphics only have to edit, compile and run iMain.cpp using Vsual C++ 6. See the listing of iMain.cpp.

```
# include "iGraphics.h"

/*
Function iDraw() is called again and again by the system.
*/
void iDraw()
{
	//place your drawing codes here
}

/*
Function iMouseMove() is called when the user presses and drags
the mouse.(mx, my) is the position where the mouse pointer is.
*/
void iMouseMove(int mx, int my)
{
	//place your codes here
}

/*
Function iMouse() is called when the user presses/releases the mouse.
(mx, my) is the position where the mouse pointer is.
*/
void iMouse(int button, int state, int mx, int my)
```

```
{
	if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		//place your codes here
	}
	if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
	{
		//place your codes here
	}
}

/*
Function iKeyboard() is called whenever the user hits a key in
keyboard. key- holds the ASCII value of the key pressed.
*/
void iKeyboard(unsigned char key)
{
	if(key == 'q')
	{
		//do something with 'q'
	}
	//place your codes for other keys here
}

/*
Function iSpecialKeyboard() is called whenver user hits special keys
like- function keys, home, end, pg up, pg down, arraows etc. you have
to use appropriate constants to detect them. A list is:
GLUT_KEY_F1,   GLUT_KEY_F2,   GLUT_KEY_F3,   GLUT_KEY_F4,   GLUT_KEY_F5,
GLUT_KEY_F6,   GLUT_KEY_F7,   GLUT_KEY_F8,   GLUT_KEY_F9,   GLUT_KEY_F10,
GLUT_KEY_F11, GLUT_KEY_F12, GLUT_KEY_LEFT, GLUT_KEY_UP, GLUT_KEY_RIGHT,
GLUT_KEY_DOWN,  GLUT_KEY_PAGE  UP,  GLUT_KEY_PAGE  DOWN,  GLUT_KEY_HOME,
GLUT_KEY_END, GLUT_KEY_INSERT.
*/
void iSpecialKeyboard(unsigned char key)
{

	if(key == GLUT_KEY_END)
	{
		exit(0);
	}
	//place your codes for other keys here
}

int main()
{
	//place your own initialization codes here before iInitialize().
	iInitialize(400, 400, "demooo");
	return 0;
}
```

Fig: iMain.cpp

3. Functions in iGraphics.h

void iInitialize(**int** width=500, **int** height=500,**char*** title="iGraphics")
Description: Creates a window of specified size and title. Lower left corner of the window is (0, 0). **Parameters:** width- Width of the window. height- Height of the window. title- Title of the window.
void iClear()
Description: Clears the screen. You should always call this function as the first line of iDraw() to avoid flickering problem (ঝিরঝির সমস্যা) while drawing. **Parameters:** none
void iSetColor(**double** r, **double** g, **double** b)
Description: Sets current drawing color. **Parameters:** r- Red component of color. A number in the range[0, 255] g- Green component of color. A number in the range[0, 255] b- Blue component of color. A number in the range[0, 255]
void iGetPixelColor **(int** x, **int** y, **int** rgb[]**)**
Description: Sets current drawing color. **Parameters:** (x, y)- co-ordinate of the pixel to check rgb[]- a 1D array of size 3 that is passed by the caller. Red, green and blue components of color are stored in this array. Note that, here you will get rgb values in the range [0, 255].
void iPoint(**double** x, **double** y, **int** size=0)
Description: Draws a point(x, y) on screen with current color. **Parameters:** x, y- Coordinates of the point. size- (Optioal)Size of the point.
void iLine(**double** x1, **double** y1, **double** x2, **double** y2)
Description: Draws a line on the screen with current color. **Parameters:** x1, y1- Coordinates of one end point. x2, y2- Coordinates of other end point.
void iCircle(**double** x, **double** y, **double** r, **int** slices=100)
Description: Draws a circle on the screen with current color. **Parameters:** x, y- Coordinates of center. r- Radius of circle. slices- (optional) Number of line segments used to draw the circle.
void iFilledCircle(**double** x, **double** y, **double** r, **int** slices=100)
Description: Draws a filled-circle on the screen with current color. **Parameters:** x, y- Coordinates of center. r- Radius of circle.

slices- (Optional) Number of line segments used to draw the circle.
void iEllipse(**double** x, **double** y, **double** a, **double** b, **int** slices=100)
Description: Draws an ellipse on the screen with current color. **Parameters:** x, y- Coordinates of center. a, b- Sizes of major and minor axes. slices- (Optional) Number of line segments used to draw the circle.
void iFilledEllipse(**double** x, **double** y, **double** a, **double** b, **int** slices=100)
Description: Draws a filled-ellipse on the screen with current color. **Parameters:** x, y- Coordinates of center. a, b- Sizes of major and minor axes. slices- (Optional) Number of line segments used to draw the circle.
void iRectangle(**double** left, **double** bottom, **double** dx, **double** dy)
Description: Draws a rectangle on the screen with current color. **Parameters:** left- x-coordinate of lower-left corner of the screen. bottom- y-coordinate of lower-left corner of the screen. dx- width of rectangle. dy- height of rectangle.
void iFilledRectangle(**double** left, **double** bottom, **double** dx, **double** dy)
Description: Draws a filled-rectangle on the screen with current color. **Parameters:** left- x-coordinate of lower-left corner of the screen. bottom- y-coordinate of lower-left corner of the screen. dx- width of rectangle. dy- height of rectangle.
void iPolygon(**double** x[], **double** y[], **int** n)
Description: Draws a polygon on the screen with current color. **Parameters:** x- x coordinates of vertices of a polygon. y- y coordinates of vertices of a polygon. n- Number of vertices.
void iFilledPolygon(**double** x[], **double** y[], **int** n)
Description: Draws a filled-polygon on the screen with current color. Give the points in circular order. Note that, concave polygons may not be drawn correctly. **Parameters:** x- x coordinates of vertices of a polygon. y- y coordinates of vertices of a polygon. n- Number of vertices.
void iText(**GLdouble** x, **GLdouble** y, **char** *str, void* font=GLUT_BITMAP_8_BY_13)
Description: Displays a string on screen. **Parameters:** x, y- coordinates of the first character of the string. str- The string to show. font- (Optional)Specifies the font type. Values could be any one of the

following- {GLUT_BITMAP_8_BY_13, GLUT_BITMAP_9_BY_15, GLUT_BITMAP_TIMES_ROMAN_10, GLUT_BITMAP_TIMES_ROMAN_24, GLUT_BITMAP_HELVETICA_10, GLUT_BITMAP_HELVETICA_12, GLUT_BITMAP_HELVETICA_18}
void iShowBMP(**int** x, **int** y, **char** filename[])
Description: Shows a 24-bit .bmp file on screen. Both the width and height of the bmp file must be an integral power of 2. **Parameters:** x, y- coordinates of lower-left corner of .bmp file. filename- The path of the .bmp file.
int iSetTimer(**int** msec, void (*f)(void))
Description: Schedules a task to perform after some pre-specified time interval. The specified function f() will be called again and again automatically after the specified time interval msec. There can be at most 10 timers in your program. Once started these timers cannot be stopped. But they can be paused and resumed. **Parameters:** msec- Time interval in mili-seconds. f- The function that will be called automatically by the system again and again after the specified time interval. **Return value:** An integer denoting the index of the created timer. It is used to pause or resume the timer afterwards.
void iPauseTimer(**int** index)
Description: Pauses the timer. The timer is de-activated. **Parameters:** index- the index of the timer that is to be paused.
void iResumeTimer(**int** index)
Description: Resumes the timer. The timer is activated again. **Parameters:** index- the index of the timer that is to be resumed.

4. Frequently asked questions.

Q1. It seems my drawing is flickering. (ঝিরঝির window problem)

Ans. Add the line iClear( ) at the beginning of iDraw( ).

Q2. Can I call drawing function inside iMouse( ), iKeyboard( ) etc?

Ans. You should call drawing functions only inside iDraw( ). To control your drawing using mouse or keyboard, follow the technique in the demo program. Suppose that, you want to move a ball when arrow key is pressed. For this, inside iSpecialKeyboard() function, you just change the co-ordinate of the center of the ball. The next time when the iDraw( ) is called automatically, it will redraw the ball in the new co-ordinate.

Q3. How can I take input from the drawing window?

Ans. There is no easy way to do so. One way is to save every character the user presses (inside iKeyboard( )). Then make a string using those saved characters once the user hits '\n'. See TextInputDemo.cpp

Q4. How to show a .bmp file whose size is not integral power of 2?

Ans. Best way is to resize the file using Paint. If you do not want to resize the .bmp then still there is a way. [Think: every integer can be written as a sum of some integers which are all powers of 2. Like, 100 = 64 + 36. So a picture with size 100 x 100 can be divided into 4 pictures with dimensions 36 x 64, 36 x 36, 64 x 64 and 36 x 64]

Q4. Where should I can iSetTimer( ) and I do not understand the parameters and return values of it? How can I control the timer?

Ans. We must call iSetTimer( ) only at the beginning of main( ) function, i.e. it should be called before calling iInitialize( ). You must supply a time interval and you must also supply a function of this type: **void any_func(void)** as a parameter to iSetTimer( ). This function will be called again and again after the predefined time interval. Once started, a timer cannot be stopped completely. It can be paused or resumed. There can be maximum 10 timers in your program. These are numbered sequentially from 0 to 9.

Download link: http://nirjon.googlepages.com/iGraphics.zip