InterAct2D Programmer's Guide (Version 2 Beta: Nov 21, 2012)

1. Overview

InterAct2D provides an event-driven interface for implementing simple interactive applications such as 2D games. The library is implemented using OpenGL and GLUT. InterAct2D hides all details of windowed graphics programming from the application programmer. The following basic functionality is provided:

- System events are captured and handed to the application program. This includes user-initiated events (keyboard and mouse) and frame timer and a display refresh.
- Graphics are drawn through a simple hierarchy of shape classes.

2. Application Structure

Communication between InterAct2D and the application occurs through the function `handleEvent()`, which must be implemented by the application. InterAct2D calls this function in response to events received from the operating system. Information about the event is stored in an event object that is passed to the application through the `handleEvent()` function.

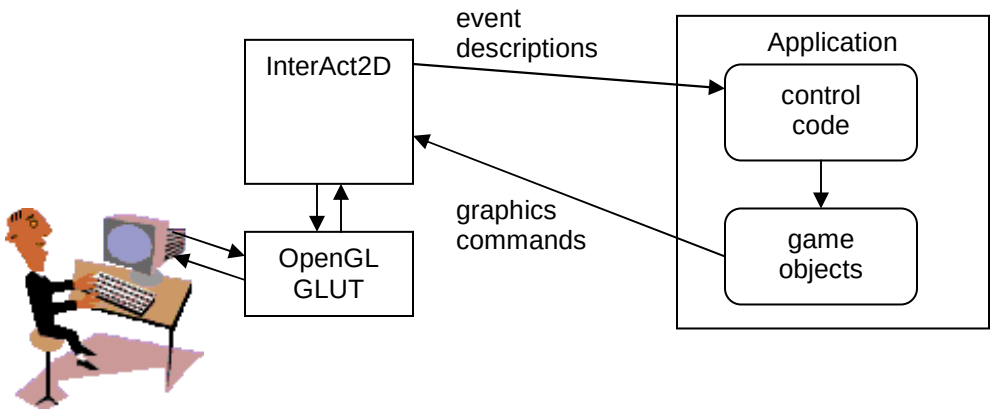
For the application programmer, it is sufficient to assume that InterAct2D is implemented as follows:

```
void main()
{
  initializeSystem();
  while (true)
  {
    Event event = waitForSystemEvents();
    handleEvent(event);
  }
}
```

Talk to OpenGL/GLUT

Talk to Application

Rather than providing a the `main()` function, the entry point into an InterAct2D application is the recurring function call: `handleEvent()`. The application is responsible for maintaining the application state between occurrences of that call. This generally means having at least one global object that is not controlled by the scope/lifetime of any function.



3. Coordinate System

The InterAct2D application window is a pixel-based coordinate system in which each point is defined by x and y coordinates. The horizontal axis is the x-coordinate increasing from zero to the right. The vertical axis is the y-coordinate, increasing from zero going down.

4. Event Handling

Class Event:

Event objects are structures consisting of a set of public attributes describing events as defined by the following table.

Event class public attributes	
`bool restart`	When true, this flag indicates that the user has pressed the restart button.
`bool repaint`	When this is true, the application should redraw all active objects.
`double elapsed_time_msec`	When positive, this value indicates that a timer event has occurred. In that case, its value is the time in milliseconds since the last timer event. The application should advance the state of all objects to reflect the passage of time.
`long screen_width`	The width of the screen in pixels.
`long screen_height`	The height of the screen in pixels.
`bool key_pressed`	When true, this flag indicates that a keyboard key has been pressed. That key's value will be stored in the attribute `key`.
`bool key_released`	When true, this flag indicates that a keyboard key has been released. That key's value will be stored in the attribute `key`.
`char key`	Value of the key that was pressed or released. This value is only valid when `key_pressed` or `key_released` is true.
`bool mouse_is_valid`	When true, this flag indicates that the mouse pointer is currently over the InterAct2D application window.
`bool mouse_moved`	When true, this flag indicates that the mouse pointer has moved to a new position over the InterAct2D application window. If this flag is true, then `mouse_is_valid` will also be true.
`Point mouse_position`	The current position of the mouse pointer. This value is only valid when `mouse_is_valid` is true.
`bool mouse_left_click`	When true, this flag indicates that the left mouse button has been pressed.
`bool mouse_right_click`	When true, this flag indicates that the right mouse button has been pressed.

Timer State

The timer provided by InterAct2D has two possible states: running and paused. If the timer is running, InterAct2D will send a timer event every 30 milliseconds (33 HZ). If the timer is paused, timer events will only be sent by user command (see the STEP command button below). Regardless of timer state, all other events will be passed to the application when they occur.

InterAct2D Control Buttons

In addition to the graphics window, InterAct2D provides a set of control buttons at the bottom of the screen. The set of buttons currently active depends on current timer state. If the timer is running the button set will be { PAUSE, RESTART, QUIT }. If the timer is paused, the button set will be { RUN, RESTART, STEP, QUIT }.

RESET	Cause a restart event to be sent to the application. It does not reset or change anything within InterAct2D itself.
PAUSE	Changes the InterAct2D timer state to paused.
RUN	Changes the InterAct2D timer state to running.
STEP	Sends a single time event with an (apparent) elapsed time of 30 msec to the application.
QUIT	Shuts down the application.

5. Graphics

Type Color: Color objects have type **`Color`**. Colors are defined by mixing red, green and blue values. A free function for creating new colors is supplied:

- **`Color(short red, short green, short blue)`** : This constructor creates a new color that is a mix of the specified intensities of red, green and blue. The parameters should be in the range of 0-255. For example **`Color(255,0,0)`** will create a bright red color and **`Color(255,255, 0)`** will create a bright yellow color.

The following colors are defined as constant values: **`white`**, **`blue`**, **`teal`**, **`green`**, **`turquoise`**, **`darkgray`**, **`brown`**, **`purple`**, **`lightblue`**, **`lightgray`**, **`gold`**, **`red`**, **`orange`**, **`pink`**, **`yellow`**, **`black`**.

Class Point: Point objects define a point in the graphics window.

constructors:

- **`Point()`** : The default constructor creates a point at (0, 0).
- **`Point(float _x, float _y)`** : This constructor creates a point at (_x, _y).
- **`Point(Point& _p)`** : The copy constructor creates a point that has the same value as the parameter.

accessors:

- **`float getX()`** : returns the current x-coordinate of the point.
- **`float getY()`** : returns the current y-coordinate of the point.

modifiers:

- **`void moveTo(Point _p)`** : Moves a point, sets it equal to the parameter _p.
- **`void move(float _dx, float _dy)`** : Moves a point by adding the parameters to the point's value. If the current point value is (x,y), the new point value will be (x+_dx, y+_dy).

Class Shape: Shape is an abstract class that is used to define the common functionality of the following classes of drawable objects: **`Oval`**, **`Rect`**, **`Line`**, **`Text`** and **`Image`**. Shape defines the following methods:

accessors:

- **`float getXSize()`** : Returns the horizontal size of the object (in pixels).
- **`float getYSize()`** : Returns the vertical size of the object (in pixels).
- **`Point getP1()`** : Returns the first point defining the object's position.
- **`Point getP2()`** : Returns the second point defining the object's position.
- **`Point getCenter()`** : Returns the center of the object in screen coordinates.
- **`Color getFillColor()`** : Returns the color used to draw the interior of the object.
- **`Color getOutlineColor()`** : Returns the color used to draw the outline/boundary of the object.
- **`short getPenWidth()`** : Returns the width used for drawing lines and outlines.
- **`void draw()`** : Causes the object to be drawn on the screen as defined by its current attributes.

modifiers:

- **`void moveTo(Point _p)`** : Moves the object such that its center is located at the point defined by _p.
- **`void move(float _dx, float _dy)`** : Moves the object by relocating it _dx pixels horizontally and _dy pixels vertically.
- **`void setFillColor(Color _color)`** : Changes the color used to draw the interior of the object.
- **`void setOutlineColor(Color _color)`** : Changes the color used to draw the outline/boundary of the object.
- **`void setPenWidth(short _width)`** : Changes the width used for drawing lines and outlines.

Additional methods supported by subclasses of Shape are as follows:

Class Oval:

constructors:

- **`Oval()`** : Creates an oval with center point (0,0), a width of zero and a height of zero.
- **`Oval(Point _center, float _x_size, float _y_size)`** : Creates an oval centered at **`_center`**, with a width of **`_x_size`** and a height of **`_y_size`**.

Class Rect:

constructors:

- **`Rect()`** : Creates a rectangle with center point (0,0), a width of zero and a height of zero.
- **`Rect(Point _center, float _x_size, float _y_size)`** : Creates a rectangle centered at **`_center`**, with a width of **`_x_size`** and a height of **`_y_size`**.

Class Line:

Lines are draw using the object's fill color. The object's outline color is not used.

constructors:

- **`Line()`** : Creates a line from point (0,0) to point (0,0).
- **`Line(Point& _p1, Point& _p2)`** : Creates a line from point **`_p1`** to point **`_p2`**.

accessors:

- **`float getLength()`** : Returns the current length of the line.

Class Text:

Text objects display character strings. The strings are C++ STL **`string`** objects. Text is drawn using the object's fill color. The object's outline color is not used. Text is drawn in a fixed size font. Each character is 9 pixels wide and 15 pixels tall. Characters are not centered in the 9x15 grid, rather they are biased towards the bottom left.

constructors:

- **`Text()`** : Creates a text object with center point (0,0) that displays an empty string.
- **`Text(Point _center, string _value)`** : Creates a text object centered at **`_center`** with a that displays the string specified by **`_value`**.

accessors:

- **`string getText()`** : Returns the string currently displayed by the text object.
- **`Point getCenter()`** : Returns the current center point for the text.

modifiers:

- **`void setText(string _value)`** : Changes the string displayed by the text object.

Class Image:

Image objects load images from files BMP files. The BMP file must use 24 bit colors. If an **`Image`** object does not have an image loaded, then it displays a 10x10 orange rectangle. This may occur if the **`Image`** object is not yet properly initialized, or if it fails to find and load the specified image.

Once color from the BMP image can be selected as a transparent color.

constructors:

- **`Image()`** : Creates an image object with center point (0,0) that displays the default orange box.
- **`Image(Point _center, string _file)`**: Creates a text object centered at **`_center`** that displays the BMP image stored in the file named **`_file`**.
- **`Image(Point _center, string _file, unsigned char _transparent_color[3])`** : Creates a text object centered at **`_center`** that displays the BMP image stored in the file named **`_file`**. Any pixels in the image that match the color specified in the third parameter are made transparent. The transparent color parameter must give red, green and blue values for the color in the range 0 to 255.
-

[VERSION 2.0 Beta. *The BMP reader used to load images occasionally fails, particularly on large images. The result will be and image of the correct size, with scrambled colors.]*

6. Random Number Generator

The **`RandomGenerator`** class is defined for generating random numbers.

constructors:

- **`RandomGenerator()`** : Creates a new object for generating random numbers.

modifiers:

- **`short random(short range) ()`** : Returns a random integer in the range [0, range-1].
- **`float frandom()`** : Returns a random real number in the range [0.0, 1.0].

7. Building InterAct2D Projects

InterAct2D projects require OpenGL and GLUT to be installed and configured for your programming environment. Include **`InterAct2D.h`** and **`InterAct2D.cpp`** in your project with an additional C++ file or files that contain your application. The minimal application code need to build a project is:

```
#include "InterAct2D.h"
void handleEvent(Event event) { }
```

This will create project that will launch a window with control buttons.

8. Sample Application

The following application creates a 50 by 50 blue rectangle in the center of the screen and moves that rectangle horizontally. The direction of motion can be changed by hitting the space bar or the left mouse button. The rectangle is reverses direction if it reaches the edge of the window.

```
// Simple application demonstrating use of InterAct2D library.
// InterAct2D Version 2.0
#include "InterAct2D.h"

////////////////////////////////////////////////////////////////////////
// Global objects to maintain state between activations of the application.
////////////////////////////////////////////////////////////////////////
Rect rectangle;
float velocity;

////////////////////////////////////////////////////////////////////////
// clearScreen(): Set screen to specified color, by drawing a big rectangle.
////////////////////////////////////////////////////////////////////////
void clearScreen(Event event, Color color)
{
	float center_x = event.screen_width/2.0f;
	float center_y = event.screen_height/2.0f;
	Rect background(Point(center_x, center_y),
			float(event.screen_width),
				float(event.screen_height));
	background.setFillColor(color);
	background.setOutlineColor(color);
	background.draw();
}

////////////////////////////////////////////////////////////////////////
// handleEvent(): Main control function called by InterAct2S for all events
////////////////////////////////////////////////////////////////////////
void handleEvent(Event event)
{
	// on restart events, reinitialize everything.
	if (event.restart)
	{
		// initial velocity is 100 pixels/sec to the right.
		velocity = 100.0;
		// reset rectangle to center of screen, 50x50 pixels, blue
		rectangle = Rect(Point(event.screen_width/2.0f,
					event.screen_height/2.0f),
				50.0f, 50.0f);
		rectangle.setFillColor(blue);
		rectangle.setOutlineColor(black);
		return;
```

```
	}

	// Handle user input from keyboard or mouse.
	if ((event.key_pressed && (event.key == ' ')) || event.mouse_left_click)
		velocity *= -1.0; // reverse velocity

	// For timer events, update state of all objects
	if (event.elapsed_time_msec > 0.0)
	{
		// motion is velocity * time
		float motion = velocity * event.elapsed_time_msec/1000.0f;
		// move the rectangle
		rectangle.move(motion, 0);
		// stop rectangle at window edges
		if (rectangle.getP1().getX() < 0.0) velocity *= -1.0;
		if (rectangle.getP2().getX() > event.screen_width) velocity *= -1.0;
	}

	// on repaint events, redraw everything
	if (event.repaint)
	{
		clearScreen(event, lightgray);
		rectangle.draw();
	}
}
```

9. Status and Known Bugs

This document applies to version 2.0 Beta of InterAct2D, released November 21, 2012. Please check the comment at the top of InterAct2D.h to verify the version number.

The following bugs are known to exist:

- The BMP reader used to load images occasionally fails, particularly on large images. The result will be and image of the correct size, with scrambled colors.

Please send bug reports to mdoherty@pacific.edu.