GUI Development: Goals

1. General GUI programming concepts
   - GUI components, layouts
   - Event-based programming
   - Graphics
   - Direct Manipulation, Animation
   - MVC architectures
   - Data-driven UIs

2. C#, .NET
   - Windows Forms
   - Events, delegates
   - GDI+
   - Threads
   - ADO.net

Goal: learn other languages quickly, same concepts
   - VB, Xwin, Java 49, ...

C# Background

- C# = VB + Java (best of both!)
- Basic statements identical to C++, Java
- Object-oriented only!
  - main) is inside a class
  - No global variables
- ‘‘interfaces’’
- No pointers (object references only), safe
- No delete: automatic garbage collection
- Error Handling, exceptions (try, catch)
- GUI: Windows Forms
- Libraries: Data structs, databases, …
- Component-based: (“assembly”) reflection
  - No .h files
- Visual Studio
- .NET: CLR, multi-language, web, XML, services, …

C# Materials

- MSDN (integrates with VS)
- VS Dynamic Help
- Books
  - MS Visual C#.NET, MS Press
    - C# language
    - Windows Forms, GDI+
- MSDN online
GUI Topics

- Components
- Events
- Graphics
- Manipulation
- Animation
- MVC

Components API

- Properties
  - Like member fields
  - Get, set
  - E.g. Button1.Text = “Press Me”
- Methods
  - Like member functions
  - Tell component to do something
  - E.g. Button1.Show()
- Events
  - Like callback functions
  - Receive notifications from component
  - E.g. Button1.Click(e)

GUI Tree Structure

Typical command line program

- Non-interactive
- Linear execution
Interactive command line program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time

```
program:
main()
{
    decl data storage;
    initialization code;
    loop
    {
        get command;
        switch(command)
        {
            command1:
            {
                code;
            }
            command2:
            {
                code;
            }
            ...
        }
    }
}
```

Typical GUI program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time
- Event callback proc

```
GUI program:
main()
{
    decl data storage;
    initialization code;
    create GUI;
    register callbacks;
    main event loop;

    Callback1() //button1 press
    {
        code;
    }
    Callback2() //button2 press
    {
        code;
    }
    ...
}
```

GUI Events

- Mouse click
- Input device

```
Window System

App1

OK
Cancel

App2

OK
Cancel
```

C# WinApp

- "delegates" = callbacks
- Function pointers

```
C# WinApp:
Class()
{
    decl data storage;
    constructor()
    {
        initialization code;
        create GUI controls;
        register callbacks;
    }
    main()
    {
        Run(new
        {
            do stuff;
            CancelBtn_click()
            {
                do different stuff;
                App2Form_click()
                {
                    do other stuff;
                }
            }
        }
    }
    callback1()
    {
        do stuff;
    }
    callback2()
    {
        do stuff;
    }
    ...
}
```

Java: Listeners
Delegates

1. Register with a control to receive events
   - Give Control a function pointer to your callback function
   - `this.button1.Click += new EventHandler(this.button1_click);`
2. Receive events from control
   - Control will call the function pointer
   - `private void button1_click(object sender, EventArgs e){
      Button1_click();
   }

Graphics

- Screen is like a painter’s canvas
- App must paint its window contents
  - GUI components paint themselves
  - Anything else: Programmer

1. How to paint?
2. When to paint?

Pixels

Coordinate System

- Upside-down Cartesian
- `Y_window = height - Y_cartesian`
Component Hierarchy

- Each component has its own subwindow
  - Subwindow = rectangular area within parent component
  - Has own coordinate system
- Clipping:
  - Can’t paint outside its subwindow
  - Can’t paint over child components

Painting Components

- Can paint any component
- Panel is good for custom graphics area

Painting in C#

1. The GDI+ graphics library:
   ```csharp
   using System.Drawing;
   ```

2. Get the “graphics context” of a component:
   ```csharp
   Graphics g = myPanel.CreateGraphics();
   ```

3. Paint in it:
   ```csharp
   g.DrawLine(pen, x1,y1, x2,y2);
   ```

Graphics Primitives

- Line (pt1,pt2)
- Lines (pt list)
- Arc (rect)
- Curves, Bezier (pt list)
- Ellipse (rect)
- Rectangle (rect)
- Polygon (pt list)
- Image (img, x,y)
- String (string, x,y)
Graphics Attributes

- Pen (for lines)
  - Color, width, dash, end caps, joins,
- Brush (for filling)
  - Color, Solid, texture, pattern, gradient
- Font, String Format (for strings)
- Bitmap/Metafile (for images)
  - Bmp, gif, jpeg, png, tiff, wmf, ...

Color

- Combinations of Red, Green, Blue
- Alpha value = opacity
- Each in [0, 255]

C#: Color.FromArgb(255, 150, 0)

Hokie Orange

Re-Paint

- Screen is like a painter’s canvas
  - All windows paint on the same surface!
  - Windows don’t “remember” what’s under them
- Need to re-paint when portions are newly exposed

- Receive Repaint events
  - Open, resize, bring to front
  - When other windows in front move, resize, close
MyApp

Open WinExp, Notepad

Close WinExplorer

Desktop gets repaint event

Repaint event sent to: Desktop, MyApp
**Repainting Static Graphics**

- Repaint event:
  - Erase (fill with background color) - usually automatically done by the control
  - Draw graphics

**In C#**

- Receive "paint" event:
  (select the event in VisStudio)
  ```csharp
  this.Paint += new PaintEventHandler(this.Form1_Paint);
  private void Form1_Paint(object sender, PaintEventArgs e)
  {  
    Graphics g = e.Graphics;
    g.DrawLine(new Pen(Color.Red,10), 10,10,300,300);
  }
  ```
- OR: Override the OnPaint method
  ```csharp
  override void OnPaint(PaintEventArgs e)
  {  
    base.OnPaint(e); //preserve base class functionality 
    Graphics g = e.Graphics;
    g.DrawLine(new Pen(Color.Red,10), 10,10,300,300); 
  }
  ```
- Can call Refresh( ) to force a repaint
Typical program structure for Dynamic Graphics

- Store data structure of graphics items
  - E.g. user drawn picture in a paint program

- Paint event:
  - Erase window (draw background color)
  - Draw graphics for items in data structure

- Other user events that alter the graphics items:
  - modify the data structure
  - send repaint event by calling Refresh()

Program Structure

C# WinApp:

```csharp
class...
    declare data storage;
    constructor(){
        initialize data storage;
    }

    cntrl1_paintEvent(e){
        draw graphics from data;
    }

    cntrl1_refesh();
```

Data structure for graphics items

- 2 approaches:
  - Store logical contents in a data structure, then draw graphics based on the data
    - E.g. drawing program: lines, shapes, colors,…
    - E.g. visualization program: data table
  - Store visual contents as an off-screen image (bitmap)
    - E.g. pixels
    - Then use g.DrawImage() in paint event

Direct Manipulation

Definition: (Shneiderman)

- Visual objects
- Selection by pointing
- Rapid, incremental, reversible actions
- Immediate, continuous feedback
Typical interaction sequence

- select item by point-n-click: **Hit testing**
  - MouseDown
- act on item by drag: **Dynamic update**
  - MouseMove
- release item
  - MouseUp

1. Hit Testing
- Mouse down, mouse over
- Which dot did user click on?
- Using components:
  - Make each dot a simple component, like a Button
  - Hit testing automatic, each component is a subwindow
  - Receive events from components, check event source
  - rectangular items, not scalable, inherit from UserControl
- Using custom graphics:
  - Get click (x,y) from MouseDown event
  - Iterate through data structure, test for hit
    - E.g: if rect.contains(x,y)
  - Data structure for fast lookup?

2. Dynamic Updating
- Dragging, stretching, …
- MouseMove events
- Using components:
  - mouseDown store x,y click in component
  - mouseMove
    - Calculate x,y delta
    - Move component by delta
- Using graphics:
  - (need to erase it, repaint other graphics, repaint new item)
  - Calculate delta, calculate new item location, store
  - Call Refresh( )
  - Draw new graphics in Paint event

Problem
- Dynamic manipulation on top of other graphics
  - Need to preserve (redraw) other graphics
  - Examples: MacDraw, powerpoint
- Simple solution:
  - Call refresh( ) or invalidate( ) while dragging
  - Paint( ) event restores other graphics
  - But: if lots of graphics, too slow & flashing!
Problem: Flashing

- Ugly flashing when repaint:
  - Paint background
  - Redraw shapes

Solution: Double buffering

- Double buffered repaint:
  - Draw all graphics in temporary off-screen image
    - Paint background color
    - Paint shapes
  - Then paint image to Window

- Bonus: C# can do this for you!
  - `Form1.DoubleBuffered = true;` // VS 2005
  - control maintains off-screen image

Rubber Band (XOR painting)

- Want multi-selection by stretching a rubber band
- Draw rubber band by inverting pixel colors
  - drawing with XOR once inverts background pixels
  - drawing with XOR twice returns to original look
    - No need to refresh(), fast!

// in mouseMove event:
// erase previous rect:  (must use screen coords, not window coords)
Control.Paint.DrawReversibleFrame(rect, Color.Black, FrameStyle.Dashed);
// update rect here based on mouse x,y
Control.Paint.DrawReversibleFrame(rect, Color.Black, FrameStyle.Dashed);
// draw new rect:
Control.Paint.DrawReversibleFrame(rect, Color.Black, FrameStyle.Dashed);
Drag-n-Drop

- Drag and Drop API for GUI controls
  - Supports data transfer
  
  ```csharp
  DestControl.AllowDrop = True;
  
  SourceControl_MouseEvent:
    this.DoDragDrop(data, DragDropEffects.Copy);
  
  DestControl_DragOver(e):
    e.Effect = DragDropEffects.Copy;
  
  DestControl_DragDrop(e):
    do something with e.Data.GetData(typeof(String));
  ```

Animation

- Update components/graphics in a loop:
  ```csharp
  for(int i =0; i<100; i++)
    button2.Left += 10;
  ```
  ```csharp
  for(int i =0; i<100; i++)
    myGraphicX += 10;
    refresh();
  ```

- but? Loops block other events.

Event-based Animation

- Use a Timer control
  - Non-visible control, fires a Tick event at specified intervals
  - Timer1.Interval = 10  //milliseconds
  - Timer1.Enabled = true  //starts animation
  - in Timer1_Tick( ) event:
    - Update graphics
    - Refresh()
  - Timer1.Enabled = false  //stops animation

- Use doublebuffering

Software Architecture  so far...

Program State
- data structures

Paint event
- display data

Interaction events
- modify data
Model-View-Controller (MVC)

**Model**
- Program State
  - data structures

**View**
- Paint event
  - display data

**Controller**
- Interaction events
  - modify data

**Advantages?**

- Multiple views for a model
  - Multi-view applications (overview-detail, brushing…)
  - Different users
  - Different UI platforms (mobile, client-side, server-side, …)
  - Alternate designs
- Multiple models
- Software re-use of parts
- Plug-n-play
- Maintenance

**Multiple Views**
E.g. C# TreeView Control

```
TreeView control

Model

TreeView control

TreeView control
```

C# Database Controls

```
DataGrid control

Model

DataSet class

```

GUI Topics

- Components
- Events
- Graphics
- Manipulation
- Animation
- MVC