

WPF 2D

(Windows Presentation Foundation)

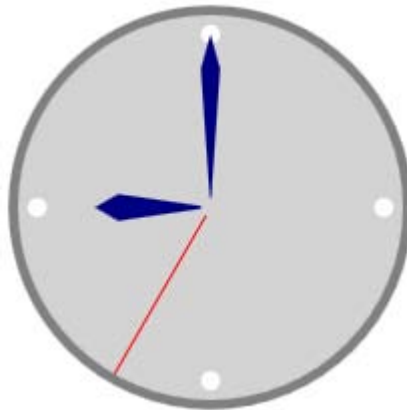
Graphics Libraries

- Raster graphics packages existed since beginning of raster displays
 - Simple Instructional Graphics Package (SGRP) in Foley, van Dam, Feiner & Hughes 2nd Ed.
 - Microsoft's Graphics Display Interface (GDI, now GDI+)
 - Java.awt.Graphics2D
 - Apple QuickDraw (based on LisaGraf for Apple® Lisa, 1980s)
- Earliest packages usually had these characteristics:
 - geometric primitives/shapes, appearance attributes specified in attribute bundles/graphical contexts/brushes, applied modally rather than in a parameter list
 - integer coordinates map directly to screen pixels on output device
 - immediate mode (no record kept of display commands)
 - no built-in functions for applying transforms to primitives
 - no built-in hierarchical modeling (no composite shapes)
- Early packages were little more than assembly languages for the display device
- Total control but program development hard

Early Graphics Library Problems (1/2)

Redrawing and Damage Repair

- Animate a clock to continuously update time
- Simple strategy
 - Regenerate entire image by redrawing all primitives
 - Inefficient for scenes composed of many primitives
- Incremental strategy
 - Update as little of the image as possible
 - Erase second hand by replacing with background color
 - Reconstruct portion of the image that was 'damaged'
 - Efficient, but far more complex to implement



- Simple solution: developer responsible for redrawing and damage repair
- Problem intrinsic to window managers that have a variety of coping strategies (e.g., backing store for images)

Early Graphics Library Problems (2/2)

Supporting User Interaction

- User clicks minute hand, location must be mapped to relevant application object
- Developer responsible for "pick correlation" (usually some kind of "point-in-bounding box rectangle" test based on pick coordinates)
- Find top-most object at clicked location
- May need to find entire composite object hierarchy from lowest-level primitive to highest level composite
 - e.g., lozenge -> hand -> clock

Supporting Multiple Display Devices

- Integer coordinates mapped to display pixels alters apparent size of image: large on low-res display & small on high-res display
- Developer responsible for coordinate-system mapping
- Application needs highly accurate internal coordinate representation (floating point for maximum accuracy)
- Map to output device by conversion factors for resolution of device

Handling Complexity

- Bookkeeping objects in scene necessary for drawing, clicking, etc.
- In order to perform operations on objects in the scene application must keep list of all primitives and their transforms
- Developer responsible for necessary data structures and logic

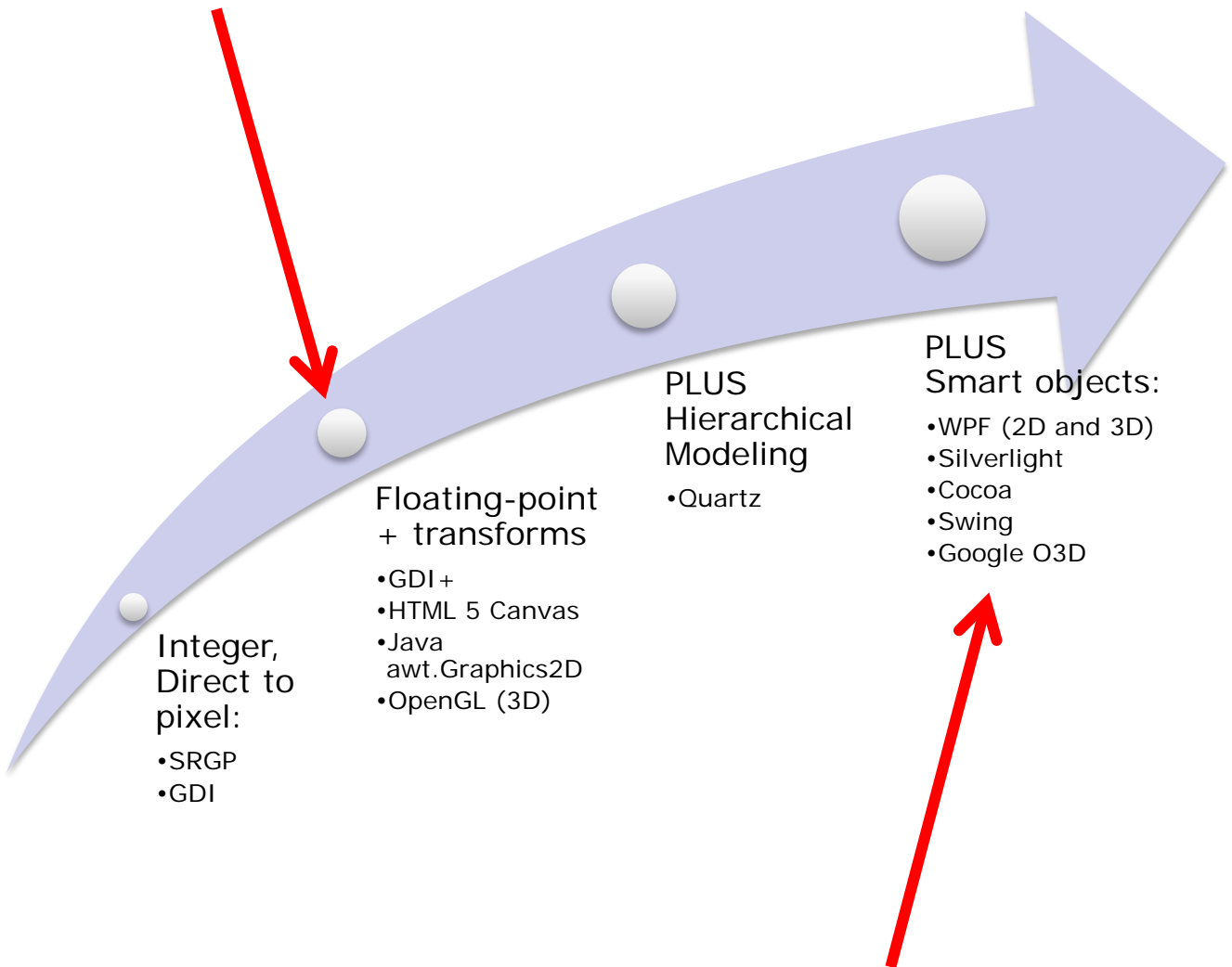
Modern Graphics Libraries (1/2)

Addressing developer burdens of older packages

- Device-Independent floating point coordinate system
 - Packages convert "model-space" to "device-space" coordinates
- Hierarchical Modeling
 - Support building scenes as hierarchy of objects, using transforms (scale, rotate, translate) to place children into their parents' coordinate systems
 - Support manipulating composites as coherent objects
- Retained Mode
 - Support respecification, redrawing, and damage-repair
 - Compiles and displays from **scene graph** (more on this later): geometric extract from general Application Model/DB maintained by the graphics library (GL)
 - Both immediate and retained mode have their uses, despite religious wars fought over which is better...
 - immediate mode: OpenGL, DirectX
 - retained mode: extensively used by game engines, WPF
- Smart Objects
 - Graphic objects have innate behaviors and interaction responses
 - e.g. Button that automatically highlights itself when cursor is over it and removes highlight when cursor moves off it
- All this provided by the package to save time and ease developing graphical applications

Modern Graphics Libraries (2/2)

Support level for the projects



Support level for the labs

WPF (1/3)

Our tool for an overview of topics in computer graphics

- Windows Presentation Foundation
 - Very high level graphics package
 - Effective for rapid prototyping
 - Available on Windows XP®, Vista, 7 only (sorry)
- Provides three layers of development:
 - Lowest: C# and Visual Basic, object-oriented API, procedural code
 - Middle: XAML, XML based declarative language (think HTML);
 - Highest: Tools/Apps for generating XAML, procedural, graphics, user interface
- C#
 - Like java: object oriented, garbage collection, very similar syntax
 - C# has Visual Studio as IDE, Java has Eclipse as IDE
 - Any WPF application can be created with just C#
 - Used for procedural code, creating new classes, etc.
- Visual Basic
 - Has access to same WPF classes as C#
 - Designed for ease of use; event-driven programming
 - Integrated with drag-and-drop GUI builder
 - Also in Visual Studio IDE
- Need only XAML in CS123, but feel free to use more...

WPF (2/3)

- XAML – eXtensible Application Markup Language
 - Based on XML - declarative 'pointy bracket' language
 - XAML similar to HTML, XML
 - HTML has fixed set of keywords you must use
 - XML is set of syntax rules for defining object in terms of elements and attributes

```

<html>
  <body>
    Hello World
  </body>
</html>

```

```

<note>
  <to Person="Andy"/>
  <from Person="Lyn"/>
  <subject>Reminder</subject>
  <body>
    You are teaching class today
  </body>
</note>

```

- XAML is a particular implementation of graphics objects with properties as set of predefined tags
- sub-elements may be declared as nested tags

```

<Canvas ID="root"
  xmlns="http://schemas.microsoft.com/2003/xaml"
  xmlns:def="Definition">
  <Button Height="50" Width="100">
    Hello World!</Button>
</Canvas>

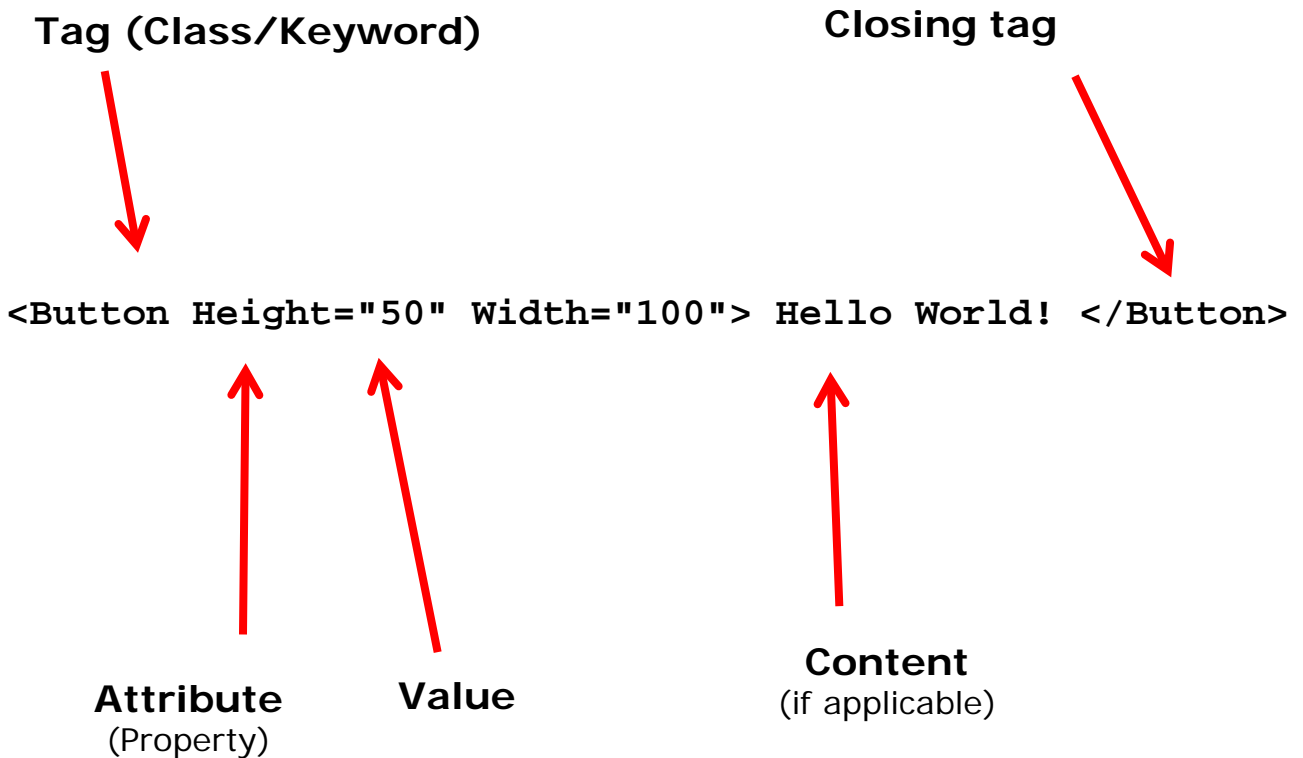
```

↖
No need to understand this,
just what makes XAML work

In case you're interested, it's called the **namespace definition** and lets the XML parser know how to parse the document.

WPF (3/3)

Anatomy of an XML Tag:



If there is no content:

```
<Button Height="50" Width="100" />
```

`/>` replaces closing tag

Full XAML More Complex

A bit of "syntactic vinegar" taught in two WPF Labs

WPF 2D Lab: (choose 1)

- Tuesday 9/15, 7-10 pm
- Friday 9/18, 7-10 pm

WPF 3D Lab: (choose 1)

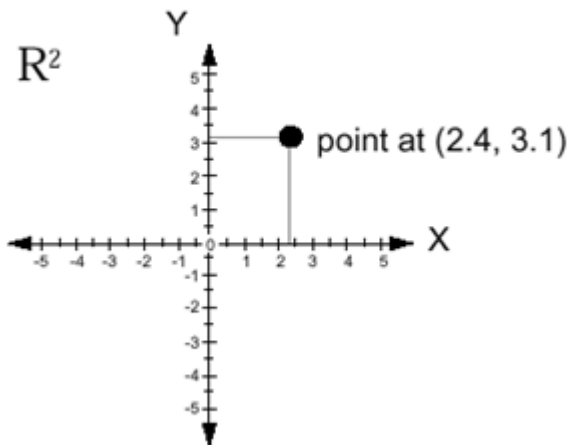
- Tuesday 9/22, 7-10 pm
- Friday 9/25, 7-10 pm

We will show rapid prototyping of simple programs that let you explore **hands-on** key topics in computer graphics:

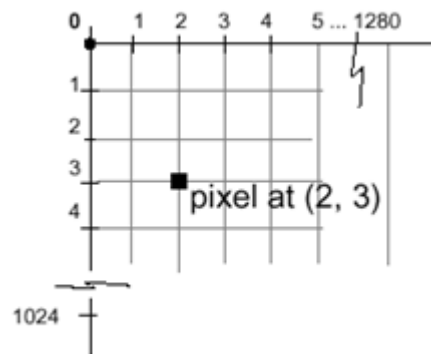
- drawing polygons, texture mapping, transformations, lighting model, camera model
- driving a simple car before building it...

Coordinate Systems (1/2)

- Cartesian coordinates (mathematics, engineering...)
 - Typically modeled as floating point
 - Typically X increasing right, Y increasing up
- Display coordinates
 - Integer only
 - Typically X increasing right, Y increasing down
 - 1 unit = 1 pixel



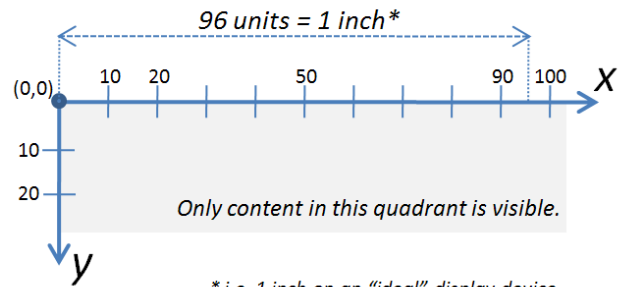
x, y Cartesian grid



Integer Raster Grid

Coordinate Systems (2/2)

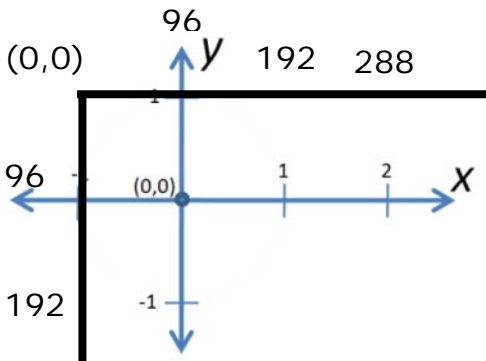
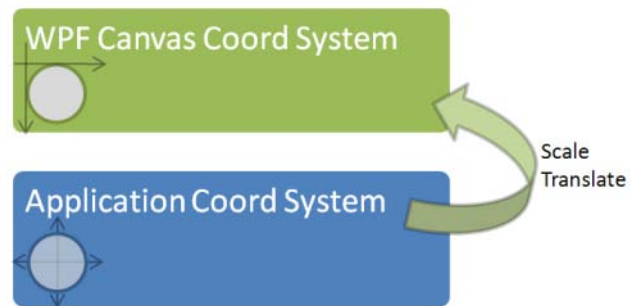
- WPF Coordinates
 - X increases right, Y increases down
 - 1 unit = 1/96th inch



* i.e. 1 inch on an "ideal" display device.

- What if you want your application's coordinate system to be:
 - 1 inch = 1unit
 - centered at (1inch, 1inch)
 - Y increasing upward?

- WPF uses transforms to change between coordinate systems
- Transforms act as conversion factor from application coord system to WPF coord system



Black=WPF Coord
Blue=Canvas Coord

```

<Canvas... >
  <RenderTransform >
    <ScaleTransform ScaleX="96" ScaleY="96" />
    <ScaleTransform ScaleX="1" ScaleY="-1" />
    <TranslateTransform X="96" Y="96" />
  </RenderTransform >
</Canvas >
    
```

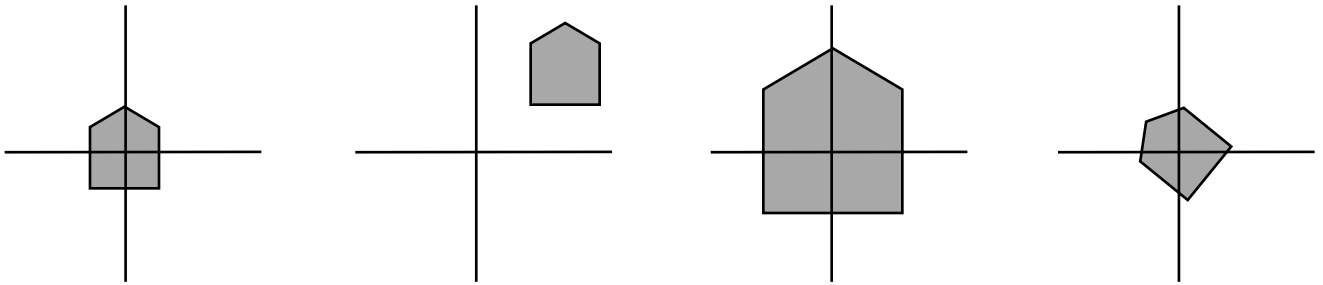
Alternatively:

```

<ScaleTransform ScaleX="96" ScaleY="-96" />
    
```

Transformations

- Geometric Transformations in 2D



Original

Translate

Scale¹Rotate¹

```
<TranslateTransform X= "40" Y="40" />
```

```
<ScaleTransform ScaleX="2" ScaleY="2" Center ... />
```

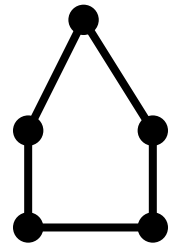
```
<RotateTransform Angle="45" Center ... />
```

- Geometric Transformations in 3D is similar
 - Translate in X,Y,Z
 - Scale in X,Y,Z
 - Rotate about X, about Y, or about Z
 - More on this later, including how to rotate and scale about arbitrary points, not just origin
 - Remember origin is top-left, not center of display

¹Transform relative to origin (0,0)

Representing Shapes

- Polygons
 - Lines drawn between ordered points
 - Same first and last point make *closed polyline* or *polygon*



<Polygon

```
Points="0,0 1,0 1,1 0.5,1.5, 0,1"
Stroke="Black" />
```

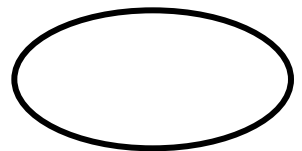
- WPF automatically closes polygons

- Special polygons

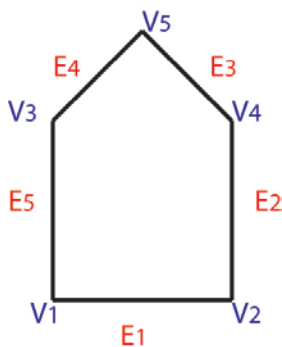
```
<Rectangle H="50" W="100".../>
```

```
<Ellipse H="50" W="100".../>
```

Height & Width refer to polygon's bounding box on major axes (x,y)



- Representation of polygon
 - Simple strategy: list of vertices and edges

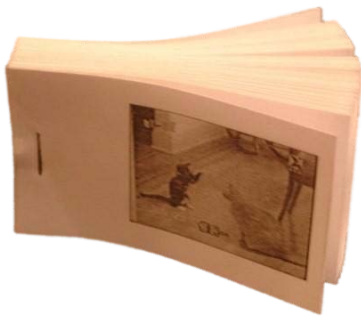


Vertexes	
1	(0,0)
2	(1,0)
3	(0,1)
4	(1,1)
5	(0.5,1.5)

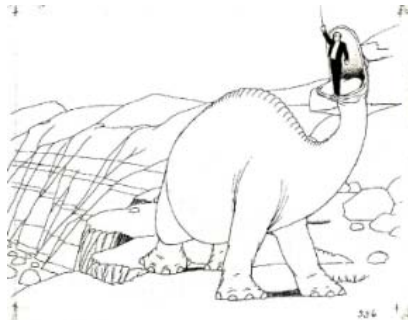
Edges	
1	(1,2)
2	(2,4)
3	(4,5)
4	(5,3)
5	(3,1)

Animation (1/2)

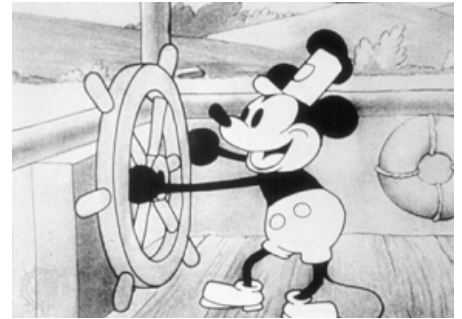
- Rapidly displaying sequence of images to create an illusion of movement
 - Flipbook (<http://www.youtube.com/watch?v=AslYxmU8xlc>)
 - Gertie the Dinosaur
 - Mickey Mouse
 - Computer: keyframe animation, procedural animation, motion capture (mocap), etc.
 - Keyframe animation: spec keyframes, computer interpolates (e.g., ball bouncing)



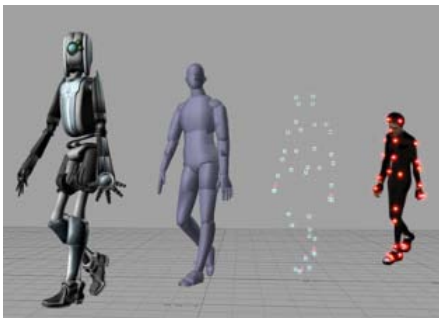
Flipbook



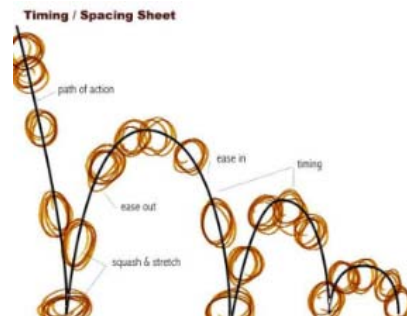
Gertie



Mickey Mouse
(Steamboat Willie)



Computer mocap



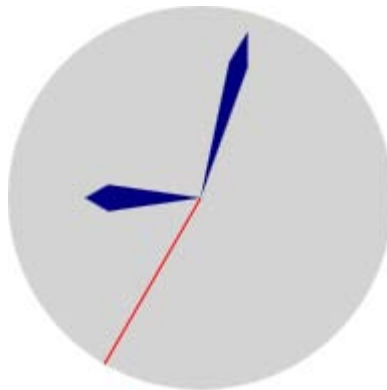
Keyframe Animation

Animation (2/2)

- Graphics packages capable of simple animation
 - Timers, Storyboards, etc.
 - Can build up to more complex animations
 - WPF can bind objects' properties to animations

```
<BeginStoryboard>
  <Storyboard>
    <DoubleAnimation
      TargetName='minuteRotation'
      TargetProperty='Angle'
      From='0.0' To='360.0'
      Duration='00:10:00'
      RepeatBehavior='Forever' />
  </Storyboard>
</BeginStoryboard>
```

Changes the value of the angle in 'minuteRotation' from 0 to 360 over 10 seconds and repeat

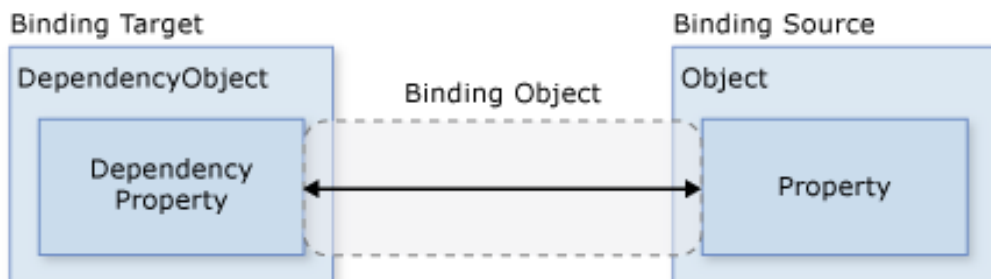


Clock Lablet

<http://www.sklardevelopment.com/graftext/ChapWPF2D/WPF2Ddemo.xbap>

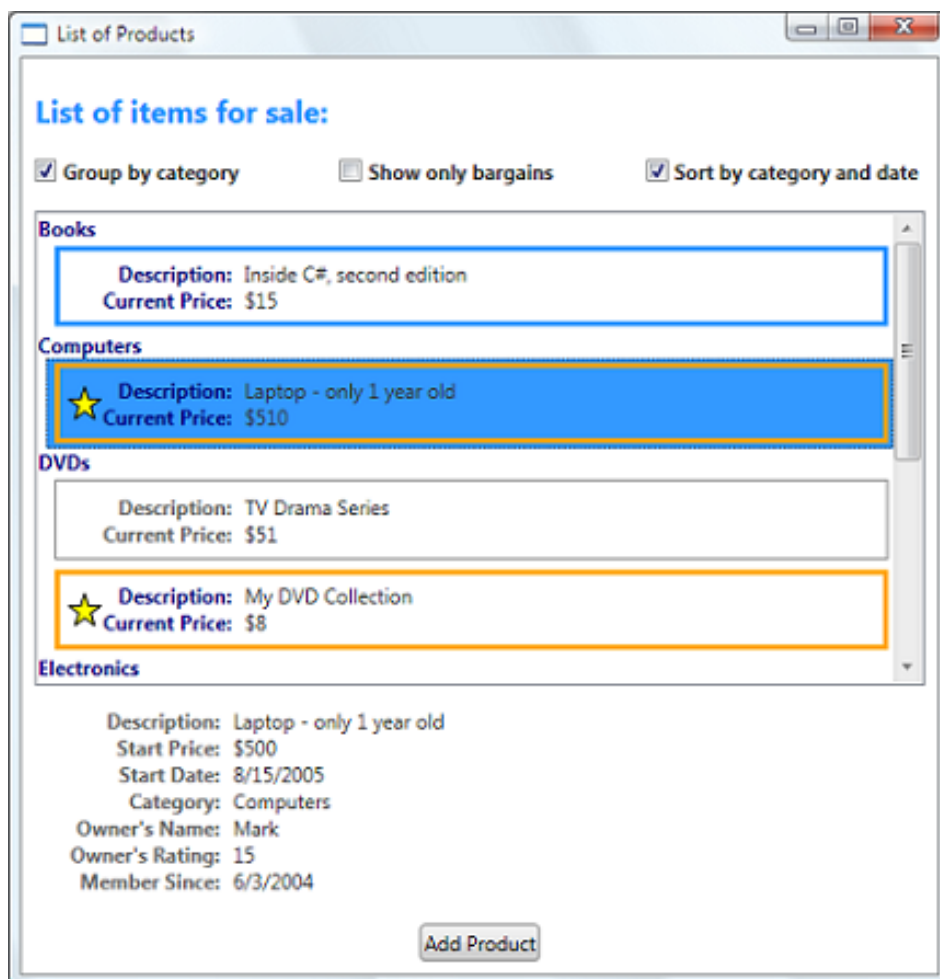
Data Binding

- Extremely powerful tool which connects the user interface with data and logic
- Once configured:
 - Elements bound to the data store are automatically populated with the correct data
 - User edits (if enabled) in the application are automatically persisted to the data store (bi-directionality); used extensively in our tablets
- Data binding's first incarnation was in business applications, where form elements are bound to database columns
- Data binding in WPF enables the separation of data (such as resources and localized strings) from the application **without the need to write any code**
- Example: Bind a slider control to control the rotation of a clock hand. As the user adjusts the slider, the rotation is updated automatically.



Data Binding Example

- Selecting item automatically lists info about item at bottom of UI
- Done entirely with Data Binding



<http://msdn.microsoft.com/en-us/library/ms752347.aspx>

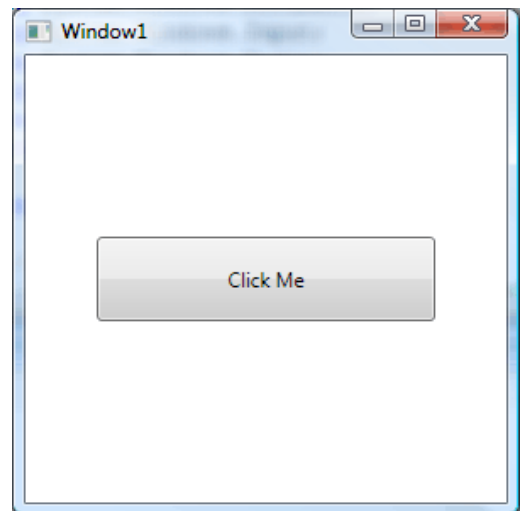
Graphical User Interface (GUI)

- WIMP: Window, Icon Menu, Pointer
 - Developed at Xerox PARC early 1970s (e.g., SmallTalk)
 - Paradigm prevalent for more than 35 years
 - Intended for ease of use for non-technical people
- Many new forms of interaction
 - Wii™ Remote, Project Natal, Multi-touch screens, etc.
- WPF GUI
 - Bind functions, data to GUI elements

XAML

```
<Button
  Name="Button1"
  Content="Click Me"
  Click="onClick1" />
```

↑
event handler

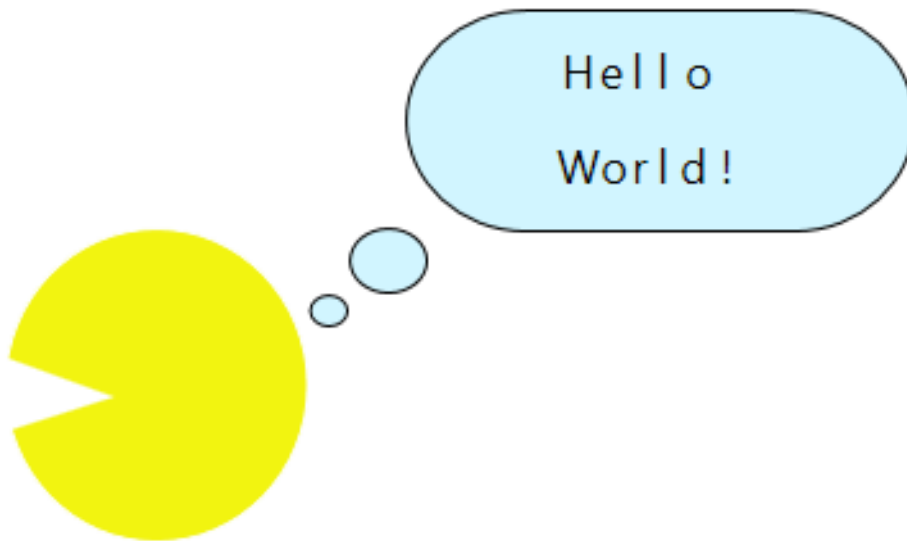


C#

```
void onClick1(object sender, RoutedEventArgs e) {
    Button1.content="Hello World";
}
```

↙ implicit invocation of a mutator

WPF Demo



By Ferdi Adeptura (fadeputr)

WPF Clock Demo (V.01)

```
<Canvas
```

```
  xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
```

```
  xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'
```

```
  Name='ClockCanvas' >
```

```
    <Ellipse
```

```
      Width='2.0' Height='2.0'
```

```
      Canvas.Left='-1.0' Canvas.Top='-1.0'
```

```
      Fill='lightgray' />
```

```
</Canvas>
```

Result: Ellipse is not visible. Why?

WPF Clock Demo (V.02)

```
<Canvas ... >
```

```
<Canvas.RenderTransform>
```

```
<ScaleTransform ScaleX='96' ScaleY='96'  
  CenterX='0' CenterY='0' />
```

```
</Canvas.RenderTransform>
```

```
<Ellipse
```

```
  Width='2.0' Height='2.0'
```

```
  Canvas.Left='-1.0' Canvas.Top='-1.0'
```

```
  Fill='lightgray' />
```

```
</Canvas>
```

Result:



WPF Clock Demo (V.03)

```
<Canvas ...>
```

```
<Canvas.RenderTransform>
```

```
<TransformGroup>
```

```
<ScaleTransform ScaleX='96' ScaleY='96'  
  CenterX='0' CenterY='0' />
```

```
<TranslateTransform X='96' Y='96' />
```

```
</TransformGroup>
```

```
</Canvas.RenderTransform>
```

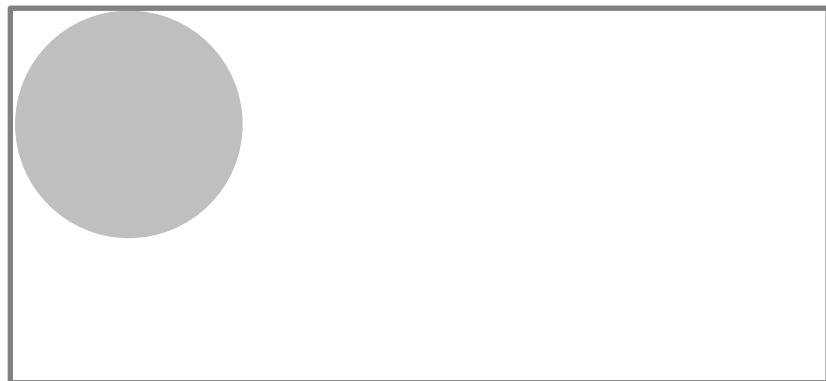
```
<Ellipse Width='2.0' Height='2.0'
```

```
  Canvas.Left='-1.0' Canvas.Top='-1.0'
```

```
  Fill='lightgray' />
```

```
</Canvas>
```

Result:



WPF Clock Demo (V.04)

```
<Canvas
```

```
  xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
```

```
  xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'
```

```
  Name='ClockCanvas' >
```

```
<!-- First, we define reusable resources, giving each a unique "key": -->
```

```
  <Canvas.Resources>
```

```
    <ControlTemplate x:Key='ClockHandStencil'>
```

```
      <Polygon
```

```
        Points='0,0 -0.2,0.8 0,1 0.2,0.8'
```

```
        Fill='Navy' />
```

```
    </ControlTemplate>
```

```
  </Canvas.Resources>
```

```
...
```

WPF Clock Demo (V.04)

`<!-- Next, we define the transform sequence applied to the canvas' content: -->`

```
<Canvas.RenderTransform>  
  <TransformGroup>  
    <ScaleTransform ScaleX='96' ScaleY='96'  
      CenterX='0' CenterY='0' />  
    <TranslateTransform X='96' Y='96' />  
  </TransformGroup>  
</Canvas.RenderTransform>
```

WPF Clock Demo (V.04)

```
<!-- NOW WE PAINT THE "SCENE" ON THE CANVAS. -->
```

```
<!-- 1. Background of the clock -->
```

```
<Ellipse
```

```
    Width='2.0' Height='2.0'
```

```
    Canvas.Left='-1.0' Canvas.Top='-1.0'
```

```
    Fill='lightgray' />
```

```
<!-- 2. The minute hand -->
```

```
<Control Name='MinuteHand'
```

```
Template='{StaticResource ClockHandStencil}' />
```

```
<!-- don't forget the closing tag -->
```

```
</Canvas>
```

Result:

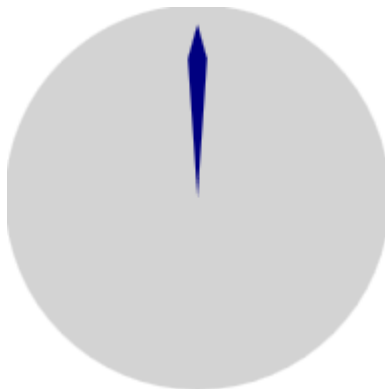


WPF Clock Demo (V.05)

Scaling down the minute hand:

```
<!-- 2. The minute hand -->
  <Control Name='MinuteHand' Template='{StaticResource
ClockHandStencil}'>
    <Control.RenderTransform>
      <TransformGroup>
        <ScaleTransform ScaleX='0.25' ScaleY='0.9'
          CenterX='0' CenterY='0' />
        <RotateTransform Angle='180'
          CenterX='0' CenterY='0' />
      </TransformGroup>
    </Control.RenderTransform>
  </Control>
```

Result:



WPF Clock Demo (V.06)

The Hour Hand

```
<!-- 3. The hour hand -->
<Control Name='HourHand' Template='{StaticResource
ClockHandStencil}'>
  <Control.RenderTransform>
    <TransformGroup>
      <ScaleTransform ScaleX='0.35' ScaleY='0.6'
        CenterX='0' CenterY='0' />
      <RotateTransform Angle='90'
        CenterX='0' CenterY='0' />
    </TransformGroup>
  </Control.RenderTransform>
</Control>
```

Result:



WPF Clock Demo (V.07)

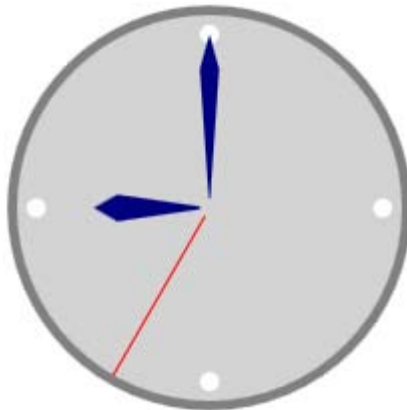
Additional customizations

```
<!-- 1(a). Background of the clock -->
<Ellipse Width='2.1' Height='2.1' Canvas.Left='-1.05'
  Canvas.Top='-1.05' Fill='gray' />
<Ellipse Width='2.0' Height='2.0' Canvas.Left='-1.0'
  Canvas.Top='-1.0' Fill='lightgray' />

<!-- 1(b). Dots at the 12/3/6/9 positions -->
<Control Template='{StaticResource CanonDot}'>
  <Control.RenderTransform>
    <TranslateTransform X='-0.05' Y='-0.95' />
  </Control.RenderTransform>
</Control>
```

...

Result:



WPF Clock Demo (V.08)

```
<!-- 2. The minute hand -->
<Control Name='MinuteHand'
Template='{StaticResource ClockHandStencil}'>

<Control.RenderTransform>
  <TransformGroup>
    <ScaleTransform ScaleX='0.25' ScaleY='0.9'
      CenterX='0' CenterY='0' />
    <RotateTransform x:Name='minuteInitializer'
      CenterX='0' CenterY='0' Angle='180' />
    <RotateTransform x:Name='minuteAnimator'
      CenterX='0' CenterY='0' Angle='0' />
  </TransformGroup>
</Control.RenderTransform>
</Control>
```



Defined in the Canvas

WPF Clock Demo (V.08)

```
<Canvas.Triggers>
<EventTrigger RoutedEvent='FrameworkElement.Loaded'>
  <EventTrigger.Actions>
    <BeginStoryboard>
      <BeginStoryboard.Storyboard>
        <Storyboard>
          <!-- THE MINUTE HAND, currently rotating
              too rapidly, for demo's sake... -->
          <DoubleAnimation
            Storyboard.TargetName='minuteAnimator'
            Storyboard.TargetProperty='Angle'
            From='0.0' To='360.0'
            Duration='00:03:00.00'
            RepeatBehavior='Forever'
          />
        </Storyboard>
      </BeginStoryboard.Storyboard>
    </BeginStoryboard>
  </EventTrigger.Actions>
</EventTrigger>
</Canvas.Triggers>
```