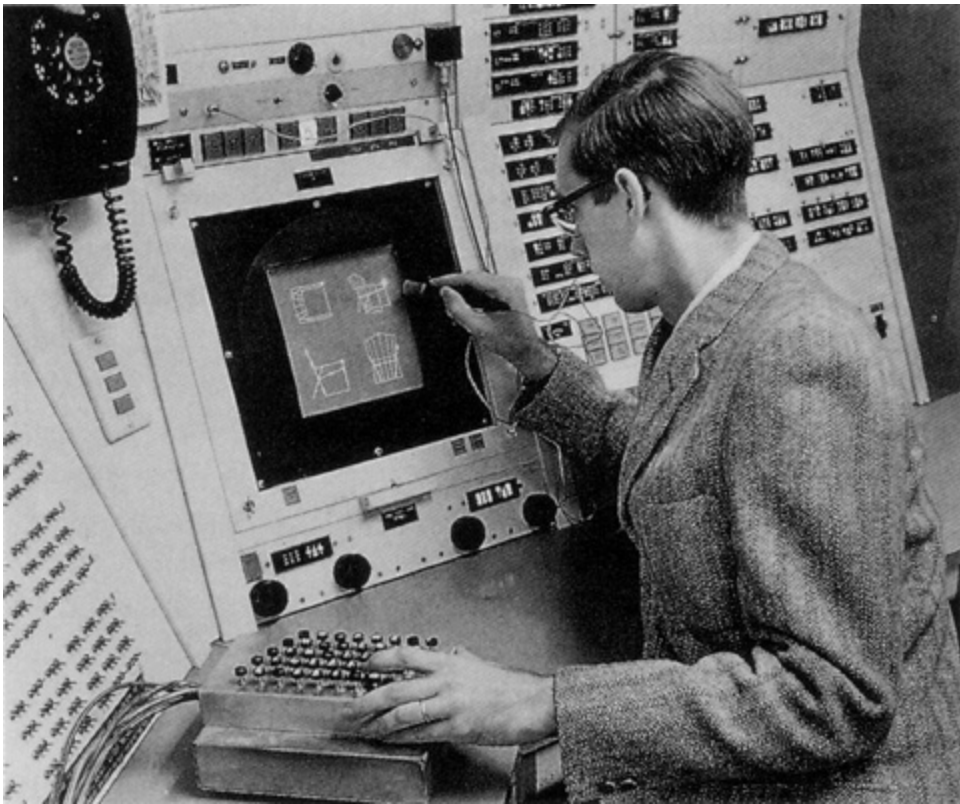


Using WPF for Computer Graphics

Matthew Jacobs, 2008

Evolution of (Graphics) Platforms 1/2

- Graphics platforms have been going through the same evolution from low-level to high-level that programming languages and development platforms have experienced



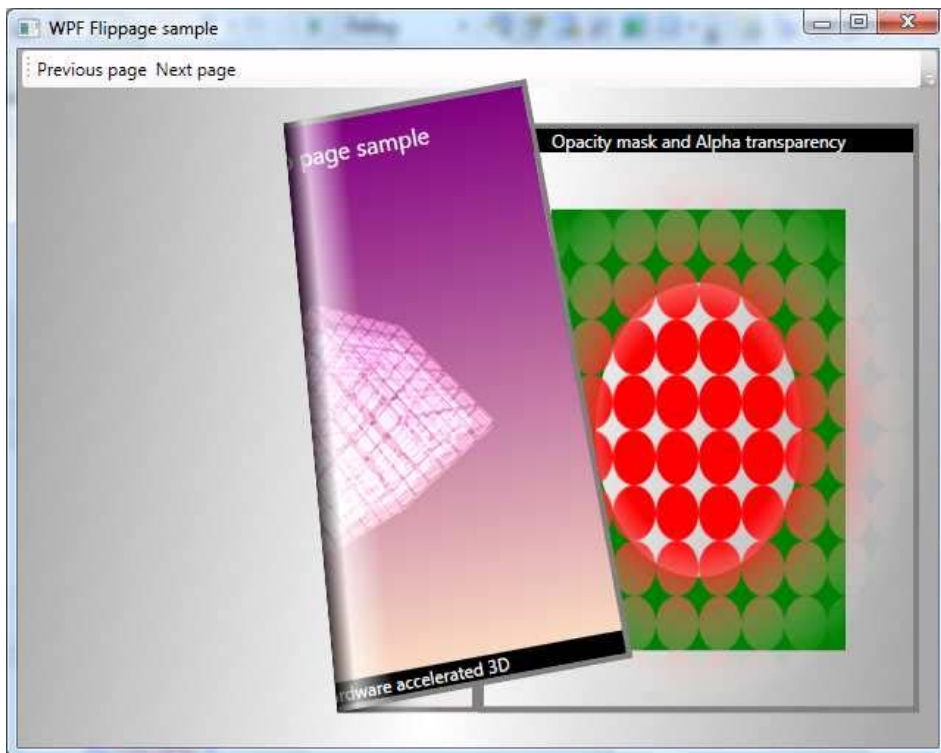
Evolution of (Graphics) Platforms 2/2

- Transfers many of the implementation details to the platform software
 - C++ → Java
 - Sketchpad → CAD
 - OpenGL → WPF3D¹
- Focus on *what is to be accomplished*, rather than on *how to accomplish it*

¹ WPF3D didn't actually evolve from OpenGL, rather, WPF3D, when compared to OpenGL, is a package that frees the programmer from a great deal of the implementation details

What is WPF?

- Windows Presentation Foundation
- **High-level** tools for building powerful Windows applications that incorporate user interface and media elements



WPF and XAML

- XAML (e**X**tensible **A**pplication **M**arkup **L**anguage) code which is
 - **high-level**
Much less code is needed
 - **declarative**
Statements *declare* objects and their relationships, different from *imperative* programming where instructions are written to *control* objects.
 - **XML-based** (pointy-brackets)

Why use WPF for graphics? (1/2)

- Built-in library support for
 - UI elements
 - Text
 - 2D Graphics
 - 3D Graphics
- Rapid prototyping
- Powerful tools
 - XAMLPad
 - Visual Studio
 - Expression Blend

Why use WPF for graphics? (2/2)

- Higher level of abstraction than OpenGL
- Declarative language (XAML)
 - Supports experimentation and rapid prototyping
 - No need for complex IDEs and lengthy compile/link cycles.
- Draw, transform, and animate 3-D graphics
- Supports 3-D coordinate spaces, cameras/projections, model/mesh primitives, texturing, illumination, transformation and animation of models

WPF and/or OpenGL?

- OpenGL is the 'assembly language' of computer graphics
- WPF contains many high-level packages for easily building graphics
- Apples to oranges!
- OpenGL is much more similar to Microsoft's Direct3D (which is the underlying engine that supports WPF)

A Few XML Pointers

- Nested “tags” contain zero or more “properties”
- Tags are specified in pointy-brackets and always contain matching closing tags (or an “empty element tag”):

```
<SomeTag> ... </SomeTag>
```

```
<AnotherTag />
```

- Tags nested within other tags are referred to as children
- XML is used to hold and carry data

Simple XML Example

```
<Note>
  <To>Andy Van Dam</To>
  <From>CS123 TA</From>
  <Message>I think the students are really
    getting it!</Message>
</Note>
```

- Could also be written as:

```
<Note To="Andy Van Dam" From="CS123
  TA" Message="I think the students
  are really getting it!" />
```

- To, From, and Message are all *child* elements of the Note element

Simple XAML Example

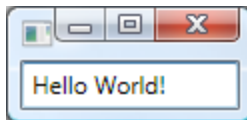
```

<Window ← Declare Window object
  x:Class="SimpleWPF.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Hello App" ← "Magic" that handles XAML XML tags, don't worry about this!
  Height="59" ← Declare window object properties
  Width="101">
  <Grid ← Declare child element Grid
    <TextBox Text="Hello World!" />
  </Grid>
</Window>

```

Parses (in XAMLPad) to the actual WPF Object it represents

Declare TextBox child of Grid object with Text property "Hi Mom"



Look Ma, No slow compile/link cycles!

Getting Started with WPF 3D (1/2)

- Need a Viewport control
 - Is a 2D control which displays a 3D scene
- To display a scene
 - Create and position a set of geometric objects (“models”)
 - Place and configure one or more lights
 - Place and configure a camera
- The viewport immediately displays a rendering of the scene
 - No need to “take a photo”; the camera is always “on”
 - No need to explicitly tell WPF to traverse the model tree, it’s all under the hood!

Getting Started with WPF 3D (2/2)

- Viewport 'stencil'

```
<Viewport3D ...>  
  <Viewport3D.Camera>  
    <PerspectiveCamera ... />  
  </Viewport3D.Camera>  
  <ModelVisual3D>  
    <Model3DGroup>  
      ...  
    </Model3DGroup>  
  </ModelVisual3D>  
</Viewport3D>
```

Set camera properties

Set up lights and objects

Setting up a Camera

...

```
<Viewport3D.Camera>  
<PerspectiveCamera  
    FarPlaneDistance="20"  
    LookDirection="5,-2,-3"  
    UpDirection="0,1,0"  
    NearPlaneDistance="1"  
    Position="-5,2,3"  
    FieldOfView="45" />  
</Viewport3D.Camera>
```

...

Adding Light to Your Scene

- Add a simple ambient light source

...

```
<Model3DGroup>  
  <AmbientLight  
    Color= '#FFFFFF' />
```

...

```
</Model3DGroup>
```

...

Placing a Triangle (1/4)

- Triangles are the only 3D primitive supported by WPF (but you know how to make others)
- Need to define a 'resource' in the application
 - Kind of like setting a variable
 - Allows programmer to reuse objects and values
- Want to define a reusable `MeshGeometry3D` which defines our first triangle

Placing a Triangle (2/4)

- Resources go in the Resources property of the parent element they belong to

```
<MeshGeometry3D
  x:Key= 'MeshPyramid'
  Positions= '0,0,75 -50,-50,0 50,-50,0'
  TriangleIndices= '0 1 2' />
```

```
<Window
  x:Class="SimpleWPF.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Hello App"
  Height="59"
  Width="101">

  <Window.Resources>
  ... ←
  </Window.Resources>

  <Viewport3D>
  ...
  </Viewport3D>
</Window>
```

Placing a Triangle (3/4)

- Now that we have a MeshGeometry3D Resource for our triangle, we can use it in our scene
- Use a GeometryModel3D which references the MeshGeometry3D resource

```
<GeometryModel3D  
    Geometry='{StaticResource  
    MeshPyramid}' />
```

- `{StaticResource rsrcname}` is a lot of magic. Just think of it as referencing the resource defined above. For more information, look up WPF data binding

Placing a Triangle (4/4)

...

```
<Viewport3D ...>
  <Viewport3D.Camera>
    <PerspectiveCamera ... />
  </Viewport3D.Camera>
  <ModelVisual3D>
    <Model3DGroup>
      <GeometryModel3D
        Geometry=' {StaticResource
          MeshPyramid}' />
    </Model3DGroup>
  </ModelVisual3D>
</Viewport3D>
```

...

Drawing a Pyramid

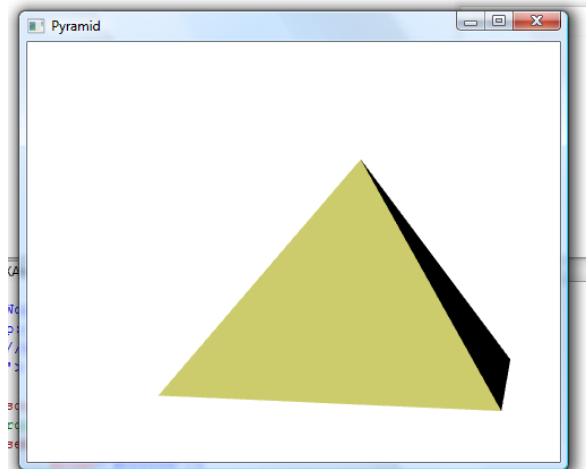
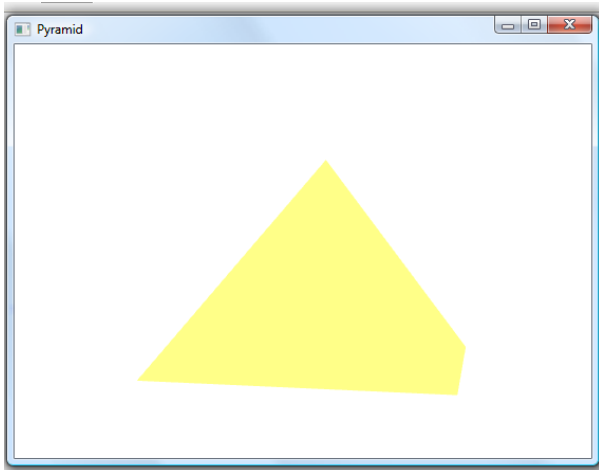
- Now that we know how to draw a triangle, we can draw more complicated shapes
- Modify the MeshGeometry3D to add more triangles to our mesh

```
<MeshGeometry3D x:Key='MeshPyramid'  
    Positions='0,0,75 -50,-50,0 50,-50,0  
              0,0,75 50,-50,0 50,50,0 0,0,75  
              50,50,0 -50,50,0 0,0,75 -50,50,0  
              -50,-50,0'  
    TriangleIndices='0 1 2 3 4 5 6 7 8 9  
                    10 11'  
  
/>
```

Improving the Pyramid

- Our pyramid doesn't quite look very good
- Need better lighting
- Use a directional light source

```
<DirectionalLight Color='#FFFFFF'  
    Direction='1, 1, -1' />
```



Working With Transformations

- We can apply transformations (T,R,S and general matrix transforms) to Model3DGroups

```
<Model3DGroup.Transform>
<RotateTransform3D
  CenterX="0" CenterY="0" CenterZ="36.5">
  <RotateTransform3D.Rotation>
    <AxisAngleRotation3D Angle="37"
      Axis="1 0 0"/>
  </RotateTransform3D.Rotation>
</RotateTransform3D>
</Model3DGroup.Transform>
```