WPF provides a wide range of 2D graphics which can be enhanced as per your application requirements. WPF supports both Drawing and Shape objects that are used for drawing graphical content.

Shapes and Drawing

* Shape class is derived from the FrameworkElement class, Shape objects can be used inside panels and most controls.
* WPF provides some basic shape objects which are derived from the Shape class such as Ellipse, Line, Path, Polygon, Polyline, and Rectangle.
* Drawing objects, on the other hand, do not derive from the FrameworkElement class and provide a lighter-weight implementation.
* Drawing objects are simpler as compared to Shape objects. They have better performance characteristics as well.

Example

Let’s take a simple example to understand how to use different shapes object.

* Create a new WPF project with the name **WPF2DGraphics**.
* The following code creates different types of shapes.

<Window x:Class = "WPF2DGraphics.MainWindow"

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

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

xmlns:d = "http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc = "http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local = "clr-namespace:WPF2DGraphics"

xmlns:PresentationOptions = "http://schemas.microsoft.com/winfx/2006/xaml/present ation/options"

mc:Ignorable = "PresentationOptions" Title = "MainWindow" Height = "400" Width = "604">

<StackPanel>

<Ellipse Width = "100" Height = "60" Name = "sample" Margin = "10">

<Ellipse.Fill>

<RadialGradientBrush>

<GradientStop Offset = "0" Color = "AliceBlue"/>

<GradientStop Offset = "1" Color = "Gray"/>

<GradientStop Offset = "2" Color = "Red"/>

</RadialGradientBrush>

</Ellipse.Fill>

</Ellipse>

<Path Stroke = "Red" StrokeThickness = "5" Data = "M 10,70 L 200,70"

Height = "42.085" Stretch = "Fill" Margin = "140.598,0,146.581,0" />

<Path Stroke = "BlueViolet" StrokeThickness = "5" Data = "M 20,100 A 100,56 42 1 0 200,10"

Height = "81.316" Stretch = "Fill" Margin = "236.325,0,211.396,0" />

<Path Fill = "LightCoral" Margin = "201.424,0,236.325,0"

Stretch = "Fill" Height = "124.929">

<Path.Data>

<PathGeometry>

<PathFigure StartPoint = "50,0" IsClosed = "True">

<LineSegment Point = "100,50"/>

<LineSegment Point = "50,100"/>

<LineSegment Point = "0,50"/>

</PathFigure>

</PathGeometry>

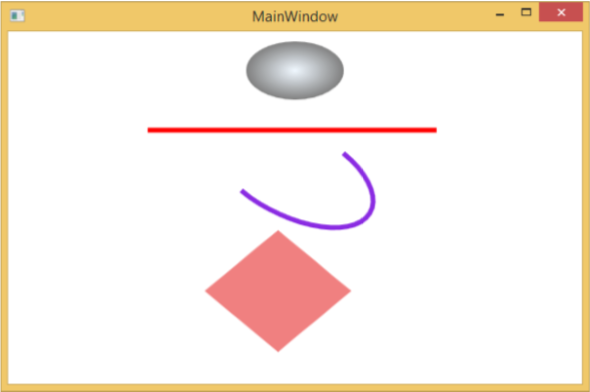
</Path.Data>

</Path>

</StackPanel>

</Window>

When you compile and execute the above code, it will produce an ellipse, a straight line, an arc, and a polygon.



Example

Let’s have a look at another example that shows how to paint an area with a drawing.

* Create a new WPF project with the name **WPF2DGraphics1**.
* The following XAML code shows how to paint different with image drawing.

<Window x:Class = "WPF2DGraphics1.MainWindow"

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

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

xmlns:d = "http://schemas.microsoft.com/expression/blend/2008"

xmlns:PresentationOptions = "http://schemas.microsoft.com/winfx/2006/xaml/present ation/options"

xmlns:mc = "http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable = "PresentationOptions"

xmlns:local = "clr-namespace:WPF2DGraphics1" Title = "MainWindow" Height = "350" Width = "604">

<Grid>

<Border BorderBrush = "Gray" BorderThickness = "1"

HorizontalAlignment = "Left" VerticalAlignment = "Top"

Margin = "20">

<Image Stretch = "None">

<Image.Source>

<DrawingImage PresentationOptions:Freeze = "True">

<DrawingImage.Drawing>

<DrawingGroup>

<ImageDrawing Rect = "300,100,300,180" ImageSource = "Images\DSC\_0104.JPG"/>

<ImageDrawing Rect = "0,100,250,100" ImageSource = "Images\DSC\_0104.JPG"/>

<ImageDrawing Rect = "150,0,25,25" ImageSource = "Images\DSC\_0104.JPG"/>

<ImageDrawing Rect = "0,0,75,75" ImageSource = "Images\DSC\_0104.JPG"/>

</DrawingGroup>

</DrawingImage.Drawing>

</DrawingImage>

</Image.Source>

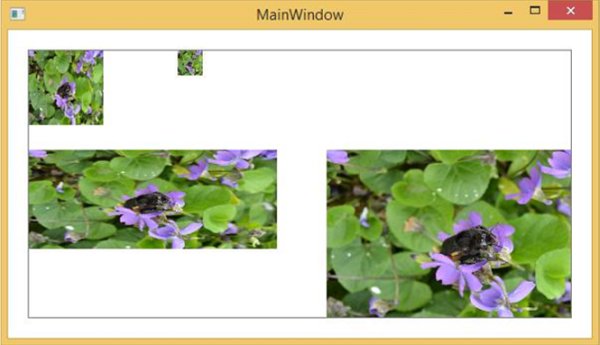
</Image>

</Border>

</Grid>

</Window>

When you run your application, it will produce the following output −



We recommend that you execute the above code and try more 2D shapes and drawings.

Windows Presentation Foundation (WPF) provides a functionality to draw, transform, and animate 3D graphics as per your application requirement. It doesn’t support full fledge 3D game development, but to some level, you can create 3D graphics.

By combining 2D and 3D graphics, you can also create rich controls, provide complex illustrations of data, or enhance the user experience of an application's interface. The Viewport3D element hosts a 3D model into our WPF application.

Example

Let’s take a simple example to understand how to use 3D graphics.

* Create a new WPF project with the name **WPF3DGraphics**.
* The following XAML code shows how to create a 2D object using in 3D geometry.

<Window x:Class = "WPF3DGraphics.MainWindow"

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

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

xmlns:mc = "http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local = "clr-namespace:WPF3DGraphics"

mc:Ignorable = "d" Title = "MainWindow" Height = "500" Width = "604">

<Grid>

<Viewport3D>

<Viewport3D.Camera>

<PerspectiveCamera Position = "2,0,10" LookDirection = "0.2,0.4,-1"

FieldOfView = "65" UpDirection = "0,1,0" />

</Viewport3D.Camera>

<ModelVisual3D>

<ModelVisual3D.Content>

<Model3DGroup>

<AmbientLight Color = "Bisque" />

<GeometryModel3D>

<GeometryModel3D.Geometry>

<MeshGeometry3D Positions = "0,0,0 0,8,0 10,0,0 8,8,0"

Normals = "0,0,1 0,0,1 0,0,1 0,0,1" TriangleIndices = "0,2,1 1,2,3"/>

</GeometryModel3D.Geometry>

<GeometryModel3D.Material>

<DiffuseMaterial Brush = "Bisque" />

</GeometryModel3D.Material>

</GeometryModel3D>

</Model3DGroup>

</ModelVisual3D.Content>

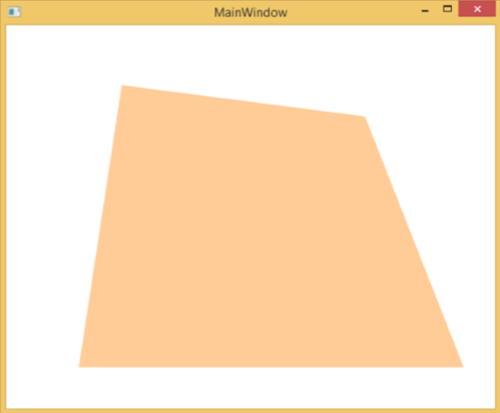
</ModelVisual3D>

</Viewport3D>

</Grid>

</Window>

When you compile and execute the above code, it will produce a 2D object in 3D.



Example

Let’s have a look at another example which shows a 3D object.

* Create a new WPF project with the name **WPF3DGraphics1**
* The following XAML code creates a 3D object and a slider. With the help of the slider, you can rotate this 3D object.

<Window x:Class = "WPF3DGraphics1.MainWindow"

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

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

xmlns:d = "http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc = "http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local = "clr-namespace:WPF3DGraphics1"

mc:Ignorable = "d" Title = "MainWindow" Height = "350" Width = "525">

<Grid>

<Viewport3D Name="viewport3D1">

<Viewport3D.Camera>

<PerspectiveCamera x:Name = "camMain" Position = "6 5 4" LookDirection = "-6 -5 -4">

</PerspectiveCamera>

</Viewport3D.Camera>

<ModelVisual3D>

<ModelVisual3D.Content>

<DirectionalLight x:Name = "dirLightMain" Direction = "-1,-1,-1">

</DirectionalLight>

</ModelVisual3D.Content>

</ModelVisual3D>

<ModelVisual3D x:Name = "MyModel">

<ModelVisual3D.Content>

<GeometryModel3D>

<GeometryModel3D.Geometry>

<MeshGeometry3D x:Name = "meshMain"

Positions = "0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1"

TriangleIndices = "2 3 1 3 1 0 7 1 3 7 5 1 6 5 7 6 4 5 6 2 0

2 0 4 2 7 3 2 6 7 0 1 5 0 5 4">

</MeshGeometry3D>

</GeometryModel3D.Geometry>

<GeometryModel3D.Material>

<DiffuseMaterial x:Name = "matDiffuseMain">

<DiffuseMaterial.Brush>

<SolidColorBrush Color = "Bisque"/>

</DiffuseMaterial.Brush>

</DiffuseMaterial>

</GeometryModel3D.Material>

</GeometryModel3D>

</ModelVisual3D.Content>

<ModelVisual3D.Transform>

<RotateTransform3D>

<RotateTransform3D.Rotation>

<AxisAngleRotation3D x:Name = "rotate" Axis = "1 2 1"/>

</RotateTransform3D.Rotation>

</RotateTransform3D>

</ModelVisual3D.Transform>

</ModelVisual3D>

</Viewport3D>

<Slider Height = "23" HorizontalAlignment = "Left"

Margin = "145,271,0,0" Name = "slider1"

VerticalAlignment = "Top" Width = "269"

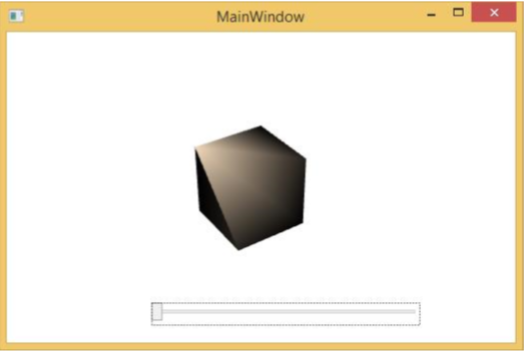
Maximum = "360"

Value = "{Binding ElementName = rotate, Path=Angle}" />

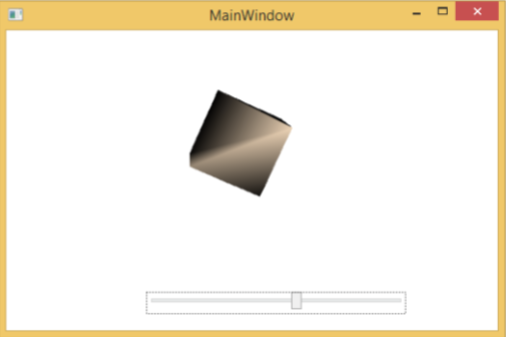
</Grid>

</Window>

When you run your application, it will produce a 3D object and a slider on your window.



When you slide the slider, the object on your window will also rotate.



We recommend that you execute the above code and try more 3D geometry.

In this article, we discuss how to create a 3D Triangle in WPF. For that follow these steps.  
  
**Step 1:** First we take a Button Control and ViewPort in our .xaml page like this:  
  
<Grid>  
        <Button Name="TriangleButton" Click="triangleButtonClick">Triangle</Button>  
        <Viewport3D Name="MainViewPort" ClipToBounds="True">  
            <Viewport3D.Camera>  
                <PerspectiveCamera  
        FarPlaneDistance="100"  
        LookDirection="-12,-11,-10"  
        UpDirection="0,1,0"  
        NearPlaneDistance="1"  
        Position="11,10,9"  
        FieldOfView="75" />  
            </Viewport3D.Camera>  
            <ModelVisual3D>  
                <ModelVisual3D.Content>  
                    <DirectionalLight  
          Color="White"  
          Direction="-2,-3,-1" />  
                </ModelVisual3D.Content>  
            </ModelVisual3D>  
        </Viewport3D>  
    </Grid>  
  
**Step 2:** After that, we take this namespace in our .cs page:  
  
using System.Windows.Media.Media3D;  
  
**Step 3:** After that, we declare a MeshPoint3D like this:

MeshGeometry3D mymesh = new MeshGeometry3D();  
  
It is used to get the Positions Collection, TriangleIndices Collection and a Normals Collection.  
  
**Step 4:** Now we create the three points of the triangle:  
  
mymesh.Positions.Add(p0);  
mymesh.Positions.Add(p1);  
mymesh.Positions.Add(p2);  
  
After that, we add the Normals and TriangleIndices in this:  
  
mymesh.TriangleIndices.Add(0);  
mymesh.TriangleIndices.Add(1);  
mymesh.TriangleIndices.Add(2);  
Vector3D Normal = CalculateTraingleNormal(p0, p1, p2);  
mymesh.Normals.Add(Normal);  
mymesh.Normals.Add(Normal);  
mymesh.Normals.Add(Normal);  
  
After that, we add the Normal Vectors, which is used for the Mesh Points like this:  
  
Vector3D Normal = CalculateTraingleNormal(p0, p1, p2);  
mymesh.Normals.Add(Normal);  
mymesh.Normals.Add(Normal);  
mymesh.Normals.Add(Normal);  
  
Here we take a function CalculateTriangleNormal; it is used to take the triangle indices, which we can get by the CrossProduct method of the Vector3D Structure.  
  
**Step 5:** After that we add a DiffuseMaterial and set its color BlueViolet like this:  
  
Material Material = new DiffuseMaterial(  
                new SolidColorBrush(Colors.BlueViolet));  
            GeometryModel3D model = new GeometryModel3D(  
                mymesh, Material);  
            Model3DGroup Group = new Model3DGroup();  
            Group.Children.Add(model);  
            return Group;  
  
**Step 6:** Now we write the function CalculateTriangleNormal:  
  
private Vector3D CalculateTraingleNormal(Point3D p0, Point3D p1, Point3D p2)  
  
{  
    Vector3D v0 = new Vector3D(  
        p1.X - p0.X, p1.Y - p0.Y, p1.Z - p0.Z);  
    Vector3D v1 = new Vector3D(  
        p2.X - p1.X, p2.Y - p1.Y, p2.Z - p1.Z);  
    return Vector3D.CrossProduct(v0, v1);  
}

**Step 7:** Now we write the code for the Click event of the Button (TriangleButton):  
  
private void triangleButtonClick(object sender, RoutedEventArgs e)

{  
    Model3DGroup triangle = new Model3DGroup();  
    Point3D p0 = new Point3D(0, 0, 0);  
    Point3D p1 = new Point3D(5, 0, 0);  
    Point3D p2 = new Point3D(5, 0, 5);  
    Point3D p3 = new Point3D(0, 0, 5);  
    Point3D p4 = new Point3D(0, 5, 0);  
    Point3D p5 = new Point3D(5, 5, 0);  
    Point3D p6 = new Point3D(5, 5, 5);

    triangle.Children.Add(CreateTriangleModel(p1, p4, p3));  
    triangle.Children.Add(CreateTriangleModel(p1, p4, p6));

    triangle.Children.Add(CreateTriangleModel(p3, p1, p6));

    ModelVisual3D Model = new ModelVisual3D();  
    Model.Content = triangle;  
    this.MainViewPort.Children.Add(Model);  
}  
  
**The Output will Be:**  
  
