PDC05
DEVELOPER POWERED

Hands-On Lab
トレーニング マニュアル

Windows Presentation Foundation での 2D と 3D のリッチ コンテンツの作成

PRSHOL08

目次

実習 1: WPF のリッチ コンテンツ

実習の目的

この実習の所要時間 : **約 60 分**

この実習の目的は、Windows Presentation Foundation の描画システムの統合と、強力な XAML を例示することです。2-D コンテンツを作成し、そのコンテンツをアニメーション化して、ビデオを追加し、最後に回転する 3-D 画面にマップします。C++、C#、または Visual Basic のコードを 1 行も記述せずに、この作業をすべて行います。

- 静的な 2-D コンテンツの作成
- アニメーションとメディアの追加
- 3-D の追加

注 : この実習の各作業の開始点を表すコードのスナップショットは、C:\Microsoft Hands-On-Lab\Creating Rich 2D and 3D Content with WPF¥Source フォルダにあります。実習を途中から始める場合や、作成中のプロジェクトが回復不可能な状態に陥った場合に、このコードが役立ちます。

実習用のコンピュータには Microsoft Windows Software Development Kit (SDK) がインストールされていて、[スタート (Start)]→[すべてのプログラム (All Programs)] メニューの [Microsoft Windows SDK] をクリックすることでドキュメントを使用できます。実習中に関連する概念の知識を深め、一般的な疑問の解答を探すため、この SDK を閉じないようにしてください。SDK には、アプリケーション プログラミング インターフェイス (API) のリファレンス マニュアルも付属しています。

演習 1 – 静的な 2-D コンテンツを作成する

この演習では、2 つの円とテキストで構成される図を作成します。**Shape** オブジェクトの輪郭を描く方法と、純色、グラデーション、ビジュアル オブジェクトで内部を塗りつぶす方法について学習します。また、背景にテキストを使用する方法についても説明します。

まず、円を作成する作業から始めます。円を描画するには、**`Ellipse`** オブジェクトを作成し、**`Width`** プロパティと **`Height`** プロパティの値が等しくなるように設定します。この例では、デバイスには依存しないピクセル値を使用して、幅 300 ピクセル、高さ 300 ピクセルの **`Ellipse`** を作成します (デバイスに依存しないピクセル、つまり DIP は、インチの 1/96 の長さです)。ブラシを使用して、この円の輪郭 (**`Stroke`**) と内部 (**`Fill`**) を描画します。

作業 1 – 円を作成する

1. Visual Studio を起動します。[スタート (Start)] ボタンをクリックし、[すべてのプログラム (All Programs)]、[Microsoft Visual Studio 2005] の順にポイントし、[Microsoft Visual Studio 2005] をクリックします。

2. [ファイル (File)] メニューの [開く (Open)] をポイントし、[プロジェクト/ソリューション (Project/Solution)] をクリックします。
3. **C:\Microsoft Hands-On-Lab\Creating Rich 2D and 3D Content with WPF¥Source¥Ex1¥Task1¥** に移動します。この実習では、作業ごとに固有の Visual Studio プロジェクトとフォルダを用意しています。
4. [RichContent.sln] ファイルを開きます。
5. [Window1.xaml] ファイルを表示します。このファイルは、**<Window>** タグとその終了タグである **</Window>** で構成され、**Window** の内部には **<Canvas>** タグと **</Canvas>** タグが含まれています。この実習では、**Window** タグの間にすべてのコンテンツを配置します。ここでは **Canvas** がそのコンテンツです。**Ctrl** キーを押しながら **F5** キーを押し、コードをコンパイルして実行します。
6. **Canvas** は **Window** 内に含まれていますが、**Canvas** の内部はまだ空です。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>
  </Canvas>
</Window>
```

7. 円を作成します。**Ellipse** オブジェクトを作成して、**Width** および **Height** の設定を同じ値にすることでこの作業を行います。**Ellipse** を表示するには、**Stroke** と **StrokeThickness** を設定します。アプリケーションを実行します。**Ellipse** は、定義済みの 2D 図形の 1 つです。**Ellipse** 以外に、**Line**、**Polygon**、**Polyline**、**Rectangle**、**Path** も簡単に使用できます。次の作業では、**Path** 図形の詳細について学習します。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>
        <Ellipse Width="300" Height="300" Stroke="Purple" StrokeThickness="20">
        </Ellipse>
  </Canvas>

</Window>
```

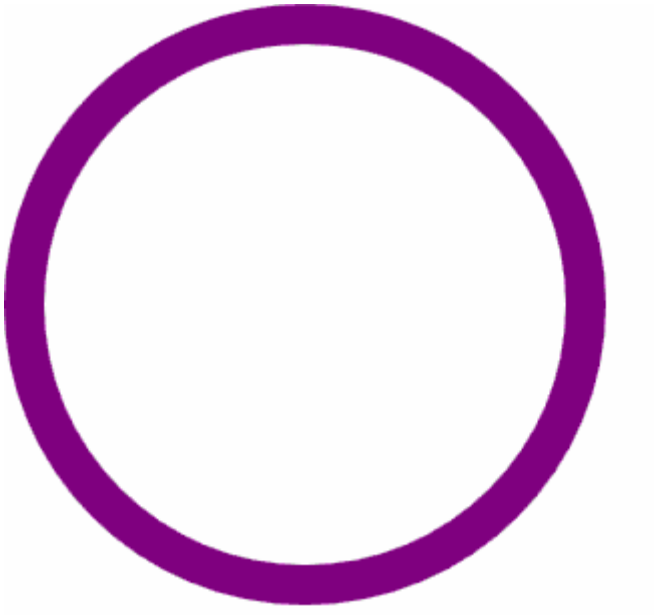


図 1: 紫色の輪郭が描かれた円

楕円の **Stroke** プロパティを "Purple" に設定して紫色の **SolidColorBrush** で楕円の輪郭を描きましたが、ここでは省略構文を使用しました。省略しないで、次のように記述することもできます。

```
<Ellipse Width="300" Height="300" StrokeThickness="20">
  <Ellipse.Stroke>
    <SolidColorBrush Color="Purple" />
  </Ellipse.Stroke>
</Ellipse>
```

WPF では、**Brush** で画面の領域を塗りつぶす方法を指示します。たとえば、**SolidColorBrush** は領域を純色で塗りつぶします。他の種類のブラシには、**LinearGradientBrush**、**RadialGradientBrush**、**ImageBrush**、**DrawingBrush**、および **VisualBrush** があります。次の手順では、**LinearGradientBrush** を使用して円の内部を塗りつぶします。

8. **LinearGradientBrush** を使用して、黄色から赤に徐々に変化する線形グラデーションによって、円の内部を塗りつぶします。LinearGradientBrush では、直線に沿ってにグラデーションを描画します。直線の始点と終点は、LinearGradientBrush の StartPoint プロパティと EndPoint プロパティで定義します。色補間の構成も線形ですが、これとは混同しないでください。すべてのグラデーションのブラシは、グラデーションの変化点の間で色を線形補間します。

   LinearGradientBrush の **StartPoint** プロパティと **EndPoint** プロパティは、”グラデーションの軸” を定義します。グラデーションではこの直線に沿って色が混合されます。**StartPoint** または **EndPoint** を指定しないと、始点と終点は既定の (0,0) と (1,1) に設定されます。この設定では、塗りつぶされる領域の左上隅から始まり、右下隅まで広がる対角線方向のグラデーションが作成されます。これらの座標は、出力領域のサイズに相対に指定することに注意してください。

   GradientStop オブジェクトを使用して、グラデーションを構成する色を指定します。各グラデーションの変化点は、色とグラデーションの軸に沿って色の位置を指定します。1 つ目

の変化点を黄色にして、**Offset** プロパティに 0 を設定します。これにより、グラデーションの変化点が、グラデーションの **StartPoint** の (0,0) に配置されます。2 つ目のグラデーションの変化点を赤色にして、**Offset** プロパティに 1を設定します。これにより、グラデーションの変化点が、グラデーションの **EndPoint** の (1,0) に配置されます。これら 2 点の間で、色が線形補間されます。これらを変更を行った後、アプリケーションを再度実行します。

```
<Ellipse Width="300" Height="300"
  Stroke="Purple" StrokeThickness="20">

  <Ellipse.Fill>
    <LinearGradientBrush>
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0" Color="Yellow" />
        <GradientStop Offset="1" Color="Red" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Ellipse.Fill>

</Ellipse>
```

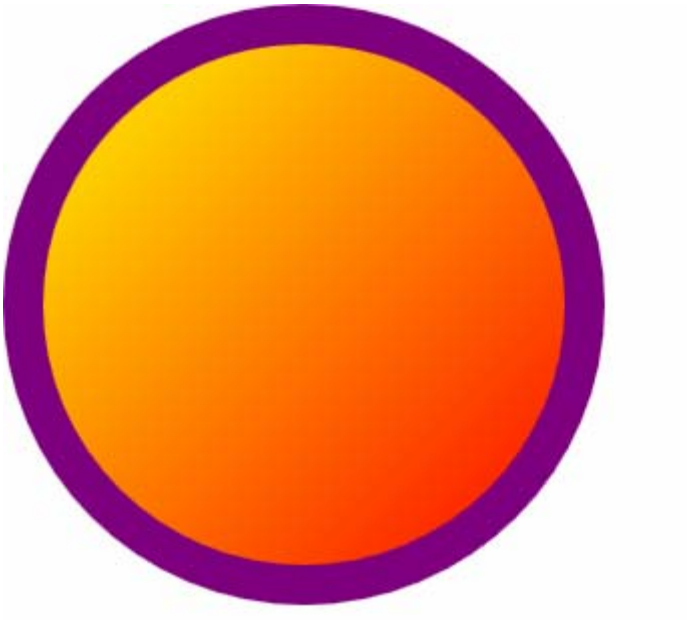


図 2: グラデーションで塗りつぶされた前の手順と同じ円

StartPoint または **EndPoint** を指定しなかったので、**LinearGradientBrush** では対角線方向にグラデーションが作成されます。**GradientStop** オブジェクトの **Offset** 値と **Color** 値だけでなく、**LinearGradientBrush** の **StartPoint** と **EndPoint** プロパティも自由に変更できます。また、追加の変化点を設定することもできます。

9. **StrokeDashArray** プロパティを “0.5,2” に設定することにで、円の輪郭が破線で描画されます。

```
<Ellipse Width="300" Height="300"
  Stroke="Purple" StrokeThickness="20"
  StrokeDashArray="0.5,2">
```

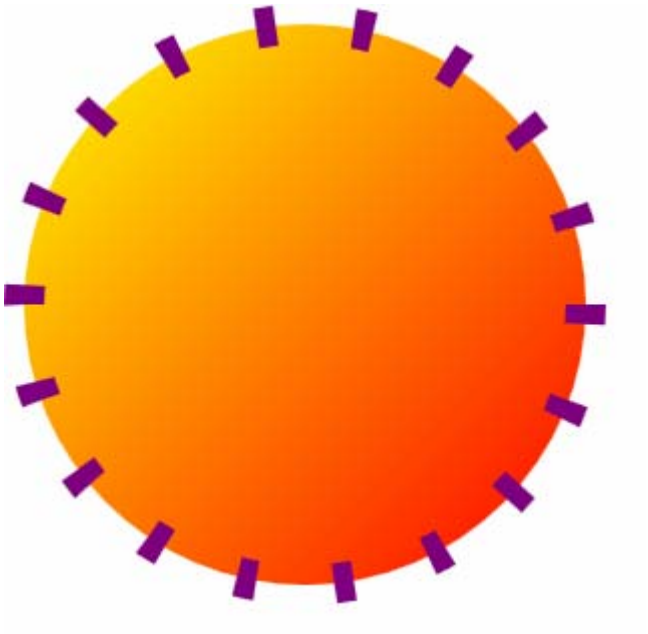


図 3: 輪郭を破線で描画した円

StrokeDashArray の設定の各値によって、図形の **StrokeThickness** に相対に破線や間隔の長さが指定されます。前のコード (**StrokeDashArray="0.5,2"**) では、**StrokeThickness** の 50% の長さの破線と、**StrokeThickness** の 2 倍の長さの間隔で構成されるパターンを作成しました。

10. 楕円の **StrokeDashCap** の値を Round に設定することで、縁に丸みを帯びた破線が描画されます。

```
<Ellipse Width="300" Height="300"
  Stroke="Purple" StrokeThickness="20"
  StrokeDashArray="0.5,2" StrokeDashCap="Round">
```

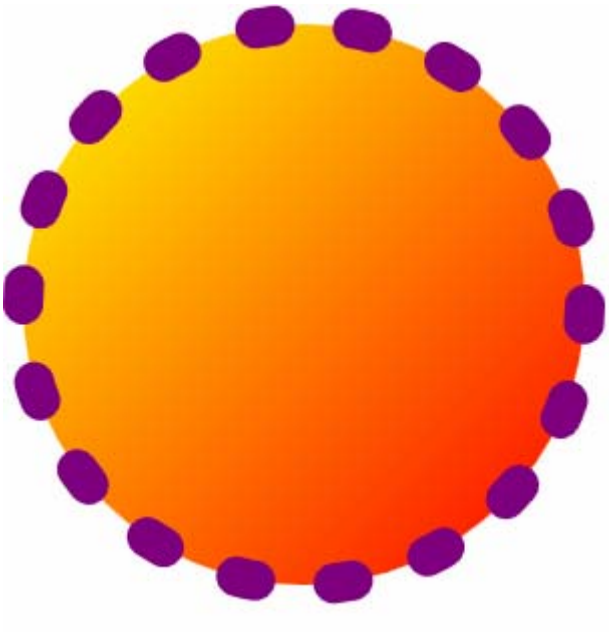


図 4: 丸みを帯びた破線で輪郭を描画した円

他の破線のスタイルには、`Flat`、`Square`、および `Triangle` があります。

11. 楕円の紫の純色の輪郭を、黒から白へ変化する線形グラデーションに置き換えます。まず、**`Path`** タグ内から **`Stroke`** プロパティを削除します。次に、**`Ellipse`** タグ内に以下の XAML を追加します。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>
    <Ellipse Width="300" Height="300"
        StrokeThickness="20"
        StrokeDashArray="0.5,2" StrokeDashCap="Round">
      <Ellipse.Fill>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Yellow" />
            <GradientStop Offset="1" Color="Red" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Ellipse.Fill>

      <Ellipse.Stroke>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Black" />
            <GradientStop Offset="1" Color="White" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Ellipse.Stroke>

    </Ellipse>
  </Canvas >
```

```
</Window>
```

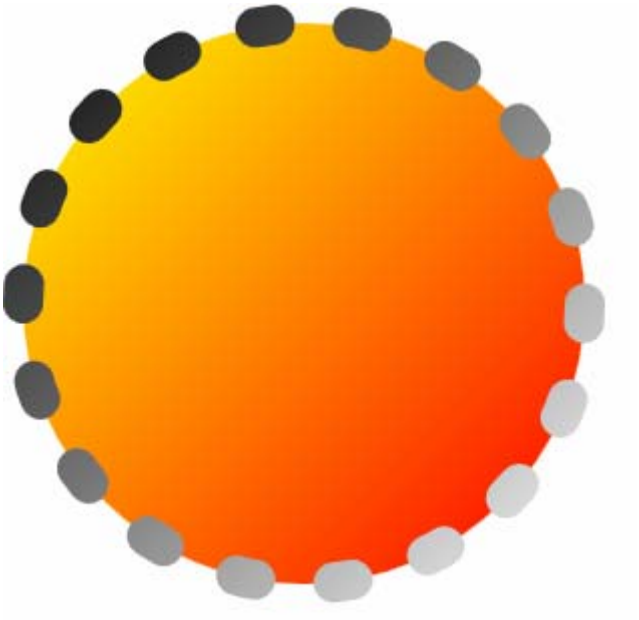


図 5: グラデーションで輪郭を描画した円

作業 2 – Path を使用してより複雑な図形を作成する

Path クラスは、定義済みの図形オブジェクト (**Line**、**Rectangle**、**Ellipse**、**Polyline**、**Polygon**) よりも汎用の図形を表します。これにより、曲線、弧、組み合わせ図形などの、複雑な図形を作成することができます。この作業では、前の作業で作成した円と同じパスを作成します。その後、より複雑な図形の一部分としてこの円を使用します。

12. **Path** を使用して、前の作業で作成した **Ellipse** を置き換えます。**Path** に、楕円のときと同じ **Stroke** と **Fill** を設定します。**Width** プロパティと **Height** プロパティに 800 を定します。**Data** プロパティを使用して、パスのコンテンツ (描画する図形) を指定します。**Data** プロパティでは **Geometry** を使用します。**Geometry** は画面領域の抽象的な指定で、表示する他のオブジェクトに依存します。円を作成するには、定義済みの図形の **EllipseGeometry** を使用できます。**EllipseGeometry** は、以前使用した定義済みの **Ellipse** の図形に似ています。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>

    <Path Width="800" Height="800"
      StrokeThickness="20" StrokeDashArray="0.5,2"
      StrokeDashCap="Round" >
```

```
      <Path.Data>
        <EllipseGeometry RadiusX="150" RadiusY="150" Center = "200,200" />
      </Path.Data>

      <Path.Fill>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Yellow" />
            <GradientStop Offset="1" Color="Red" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Path.Fill>

      <Path.Stroke>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Black" />
            <GradientStop Offset="1" Color="White" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Path.Stroke>
    </Path>

  </Canvas>

</Window>
```

図 6: Path を使用して作成した円

最初の作業で作成した **Ellipse** と同じ **Path** を作成しました。ここで、**CombinedGeometry** を使用して組み合わせ図形を作成することによって、パスの機能の利点を活用しより複雑な図形を作成します。

CombinedGeometry クラスでは、2 つの入力図形と組み合わせモードを使用して、1 つの図形を作成します。**GeometryCombineMode** プロパティを Union、Intersect、Exclude、または XOR に設定することによって、**CombinedGeometry** でどのように図形

が組み合わせられるかを制御できます。**CombinedGeometry** 自体は **Geometry** なので、**Path** 要素の **Data** プロパティを設定するのに使用できます。

13. Union 組み合わせモードを使用して、2 つの **EllipseGeometry** オブジェクトを組み合わせる **CombinedGeometry** を作成します。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>

    <Path Width="800" Height="800"
      StrokeThickness="20" StrokeDashArray="0.5,2"
      StrokeDashCap="Round" >

      <Path.Data>
        <CombinedGeometry GeometryCombineMode="Union">
          <CombinedGeometry.Geometry1>
            <EllipseGeometry Center="200,200" RadiusX="150" RadiusY="150" />
          </CombinedGeometry.Geometry1>
          <CombinedGeometry.Geometry2>
            <EllipseGeometry Center="400,200" RadiusX="150" RadiusY="150" />
          </CombinedGeometry.Geometry2>
        </CombinedGeometry>
      </Path.Data>


      <Path.Fill>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Yellow" />
            <GradientStop Offset="1" Color="Red" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Path.Fill>

      <Path.Stroke>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Black" />
            <GradientStop Offset="1" Color="White" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Path.Stroke>
    </Path>

  </Canvas>

</Window>
```

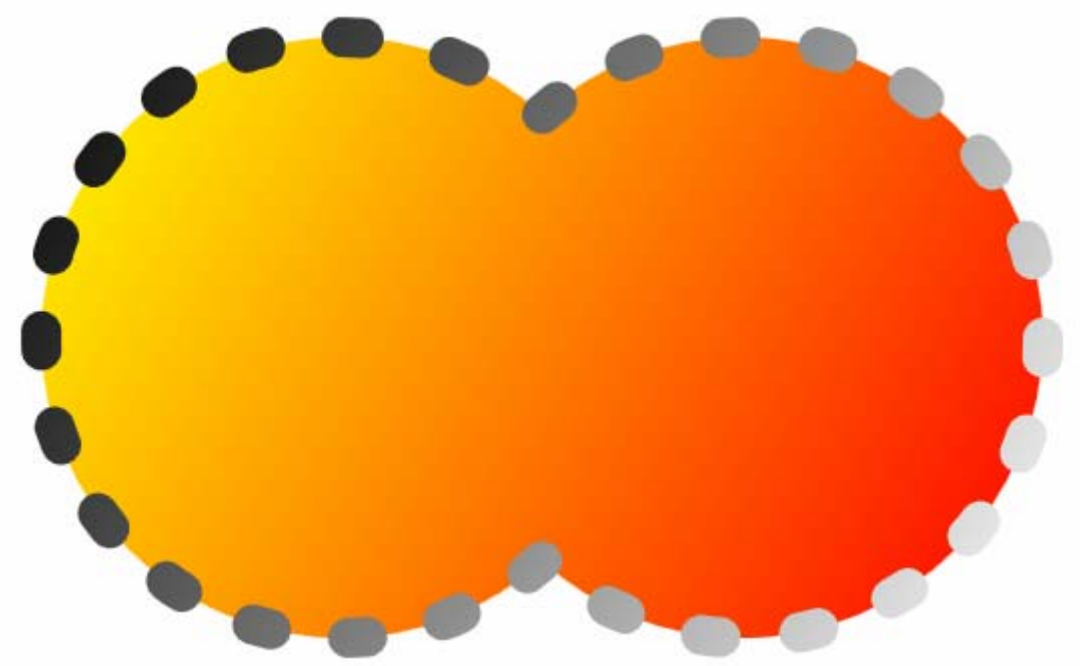


図 7: 組み合わせ図形

ここまでは、純色で領域を塗りつぶすには **SolidColorBrush** を、線形グラデーションで塗りつぶすには **LinearGradientBrush** を使用してきました。ここからは **VisualBrush** を使用して、**Visual** によって領域を塗りつぶします。**Visual** オブジェクトには多くの種類があります。**Visual** オブジェクトの種類には、コントロール、パネル、およびシェイプがあります。**Visual** オブジェクトは数と種類が豊富なので、最も強力で多目的な Windows Presentation Foundation クラスの 1 つです。

次の手順では、**VisualBrush** を使用して、テキストで **Path** の内部を描画します。

14. **VisualBrush** を作成して、パスの内部を描画するためにこれを使用します。**VisualBrush** のコンテンツを指定するには、**Visual** プロパティを設定します。**Visual** 型の入力を受け取るため、**Visual** プロパティと名付けられています。**TextBlock** を **Visual** プロパティに設定します。**TextBlock** に、以前と同様の黄色から赤へ変化する線形グラデーションの背景を設定します。必要に応じて、**TextBlock** のテキストや色を変更します。

```
<Path.Fill>
```

```
  <VisualBrush>
    <VisualBrush.Visual>
      <TextBlock Text="Hello World" FontWeight="Bold" Padding="10">
        <TextBlock.Background>
          <LinearGradientBrush>
            <LinearGradientBrush.GradientStops>
              <GradientStop Offset="0" Color="Yellow" />
              <GradientStop Offset="1" Color="Red" />
            </LinearGradientBrush.GradientStops>
          </LinearGradientBrush>
        </TextBlock.Background>
      </TextBlock>
```

```
    </VisualBrush.Visual>
  </VisualBrush>

</Path.Fill>
```

図 8: VisualBrush で塗りつぶしたパス

VisualBrush には、他にもここで紹介していない機能があります。たとえば、**Viewport** の値を `"0,0,0.1,0.1"` に、**TileMode** の値を `Tile` に設定することによって、図形の内部のテキストからパターンを作成することができます。

完成した演習 1 のコード

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>

    <Path Width="800" Height="800"
      StrokeThickness="20" StrokeDashArray="0.5,2"
      StrokeDashCap="Round" >

      <Path.Data>
        <CombinedGeometry GeometryCombineMode="Union">
          <CombinedGeometry.Geometry1>
            <EllipseGeometry Center="200,200" RadiusX="150" RadiusY="150" />
          </CombinedGeometry.Geometry1>
          <CombinedGeometry.Geometry2>
            <EllipseGeometry Center="400,200" RadiusX="150" RadiusY="150" />
          </CombinedGeometry.Geometry2>
```

```
        </CombinedGeometry>
      </Path.Data>


      <Path.Fill>

        <VisualBrush>
          <VisualBrush.Visual>
            <TextBlock Text="Hello World" FontWeight="Bold" Padding="10">
              <TextBlock.Background>
                <LinearGradientBrush>
                  <LinearGradientBrush.GradientStops>
                    <GradientStop Offset="0" Color="Yellow" />
                    <GradientStop Offset="1" Color="Red" />
                  </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
              </TextBlock.Background>
            </TextBlock>
          </VisualBrush.Visual>
        </VisualBrush>

      </Path.Fill>


      <Path.Stroke>
        <LinearGradientBrush>
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0" Color="Black" />
            <GradientStop Offset="1" Color="White" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
      </Path.Stroke>
    </Path>

  </Canvas>

</Window>
```

演習 2 – アニメーションとメディアを追加する

アニメーションとメディアを追加することによって、生き生きとした作品を作成することができます。

Windows Presentation Foundation には、統合アニメーション システムが用意されています。オブジェクトをアニメーションにするには、オブジェクトのプロパティにアニメーションを適用します。さまざまなプロパティの種類として、さまざまなアニメーションの種類があります。**Color** 値を指定するプロパティ **ColorAnimation**、double 型の値を指定するプロパティ **DoubleAnimation**、**Point** 値を指定するプロパティ **PointAnimation** などです。**EllipseGeometry** を画面中で移動させるには、**PointAnimation** に **Center** プロパティを適用します。**Rectangle** の幅をアニメーション化するには、**DoubleAnimation** に **Width** プロパティを適用します。

アニメーションは、**Storyboard** 内に保持されます。つまり、**Storyboard** は、アニメーションを編成および適用するオブジェクトです。どのオブジェクトをアニメーションにするかを指示するには、付加プロパティの **Storyboard.TargetName** を使用します。アニメーション可能にするオブジェクトには、名前を付ける必要があります。オブジェクトがフレームワーク要素の場合は、

Name プロパティに設定することで名前を割り当てます。オブジェクトがフレームワーク要素以外の場合は、x:Name マークアップ拡張を使用して名前を割り当てます。どのプロパティをアニメーションにするかを指示するには、付加プロパティの **Storyboard.TargetProperty** を使用します。

アニメーションはイベントによってトリガされます。特定のイベントが発生したときにアニメーションを再生するよう設定するには、**EventTrigger** オブジェクトを使用します。**RoutedEvent** プロパティを設定することで、どのイベントによって **EventTrigger** がトリガされるかを制御します。**EventTrigger** がトリガされたとき、これに関連付けたすべてのアクションが実行されます。アニメーションを開始するには、**BeginStoryboard** アクションを使用します。

作業 1 – アニメーションを追加する

1. 前の例で作成した破線のパターンが、パス上を回転するようにします。パスの **StrokeDashOffset** プロパティをアニメーションにすることによって、この作業を行います。**StrokeDashOffset** プロパティでは double 値を受け取るので、**DoubleAnimation** を使用します。**DoubleAnimation** をプロパティの現在値 (この例では設定されていないので 0 です) から 20 にしてアニメーションにするには、**To** プロパティに 20 を設定します。**Duration** を 10 秒に設定し、無期限に繰り返すようにします。パスが読み込まれたらすぐに再生を開始するようにします。アニメーションの対象となるように、パスに名前を付けることを忘れないでください。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>

    <Path Name="MyPath" Width="800" Height="800"
     StrokeThickness="20" StrokeDashArray="0.5,2"
     StrokeDashCap="Round" >

      <Path.Data>
        <CombinedGeometry GeometryCombineMode="Union">
          <CombinedGeometry.Geometry1>
            <EllipseGeometry Center="200,200" RadiusX="150" RadiusY="150" />
          </CombinedGeometry.Geometry1>
          <CombinedGeometry.Geometry2>
            <EllipseGeometry Center="400,200" RadiusX="150" RadiusY="150" />
          </CombinedGeometry.Geometry2>
        </CombinedGeometry>
      </Path.Data>

      <Path.Fill>
        <VisualBrush>
          <VisualBrush.Visual>
            <TextBlock Text="Hello World" FontWeight="Bold" Padding="10">
              <TextBlock.Background>
                <LinearGradientBrush>
                  <LinearGradientBrush.GradientStops>
                    <GradientStop Offset="0" Color="Yellow" />
                    <GradientStop Offset="1" Color="Red" />
```

```
                </LinearGradientBrush.GradientStops>
              </LinearGradientBrush>
            </TextBlock.Background>
          </TextBlock>
        </VisualBrush.Visual>
      </VisualBrush>
    </Path.Fill>

    <Path.Stroke>
      <LinearGradientBrush>
        <LinearGradientBrush.GradientStops>
          <GradientStop Offset="0" Color="Black" />
          <GradientStop Offset="1" Color="White" />
        </LinearGradientBrush.GradientStops>
      </LinearGradientBrush>
    </Path.Stroke>

    <Path.Triggers>
      <EventTrigger RoutedEvent="Path.Loaded">
        <EventTrigger.Actions>
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation
                Storyboard.TargetName="MyPath"
                Storyboard.TargetProperty="StrokeDashOffset"
                To="20" Duration="0:0:10"
        RepeatBehavior="Forever" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger.Actions>
      </EventTrigger>
    </Path.Triggers>

  </Path >

 </Canvas >

</Window>
```

2 つの図形が結合したものが 1 つの図形であることを、システムがどのように認識するかに注目してください。アニメーションにされた破線が 2 つの図形の間をシームレスに動きます。**Duration** プロパティを **DoubleAnimation** に変更して、アニメーションの速度を変更してみてください。

2. 次に、アニメーションにされた図形をより目立たせるために、**EllipseGeometry** オブジェクトの 1 つが画面中に飛び回わるようにします。これを行うには、**PointAnimation** を使用して、**Center** プロパティをアニメーションにします。3 秒かかって (150,100) から (650,200) に移動し、そのアニメーションを無期限に繰り返すようにします。アニメーションの対象となるように、**EllipseGeometry** に名前を割り当てます。パスが読み込まれたら、アニメーションが開始されるように設定します。

```
<Path.Data>
  <CombinedGeometry GeometryCombineMode="Union">
    <CombinedGeometry.Geometry1>
```

```
      <EllipseGeometry x:Name="MyEllipseGeometry"
        Center="200,200" RadiusX="150" RadiusY="150" />
    </CombinedGeometry.Geometry1>
    <CombinedGeometry.Geometry2>
      <EllipseGeometry Center="400,200" RadiusX="150" RadiusY="150" />
    </CombinedGeometry.Geometry2>
  </CombinedGeometry>
</Path.Data>

<Path.Triggers>
  <EventTrigger RoutedEvent="Path.Loaded">
    <EventTrigger.Actions>
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation
            Storyboard.TargetName="MyPath"
            Storyboard.TargetProperty="StrokeDashOffset"
            To="20" Duration="0:0:10"
            RepeatBehavior="Forever" />
        </Storyboard>
      </BeginStoryboard>

      <BeginStoryboard>
        <Storyboard>
          <PointAnimation
            Storyboard.TargetName="MyEllipseGeometry"
            Storyboard.TargetProperty="Center"
            From="150,100" To="650,200" Duration="0:0:3"
            RepeatBehavior="Forever" AutoReverse="True"/>
        </Storyboard>
      </BeginStoryboard>

    </EventTrigger.Actions>
  </EventTrigger>
</Path.Triggers>
```

図 9: アニメーション化されたパス

作業 2 – ビデオを追加する

1. 前の手順では、**LinearGradientBrush** を作成し、これを使用して **TextBlock** の背景を塗りつぶしました。**LinearGradientBrush** によって、**TextBlock** の **Background** ではなく **Forground** が塗りつぶされるように、コードを変更します。

```
<TextBlock Text="Hello World" FontWeight="Bold" Padding="10">

  <TextBlock.Foreground>
    <LinearGradientBrush>
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0" Color="Yellow" />
        <GradientStop Offset="1" Color="Red" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </TextBlock.Foreground>

</TextBlock>
```

2. 今度は **TextBlock** の背景に、グラデーションではなくビデオを使用します。**MediaElement** を含む **VisualBrush** を作成することで、この作業を行います。**MediaElement** は、ビデオを再生できる種類の要素です。**MediaElement** の Source プロパティは、再生するメディア ファイルの URI に設定します。

```
<TextBlock Text="Hello World" FontWeight="Bold" Padding="10">

  <TextBlock.Foreground>
    <LinearGradientBrush>
```

```
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0" Color="Yellow" />
        <GradientStop Offset="1" Color="Red" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </TextBlock.Foreground>

  <TextBlock.Background>
    <VisualBrush>
      <VisualBrush.Visual>
        <MediaElement Source="C:\Microsoft Hands-On-Lab\Creating Rich 2D and 3D
Content with WPF\Source\apollo11_launchclip04.mpg=" />
      </VisualBrush.Visual>
    </VisualBrush>
  </TextBlock.Background>

</TextBlock>
```

完成した演習 **2** のコード

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

  <Canvas>

    <Path Name="MyPath"
      Width="800" Height="800"
      StrokeThickness="20" StrokeDashArray="0.5,2"
    StrokeDashCap="Round" >
```

```
<Path.Data>
  <CombinedGeometry GeometryCombineMode="Union">
    <CombinedGeometry.Geometry1>
      <EllipseGeometry x:Name="MyEllipseGeometry"
        Center="200,200" RadiusX="150" RadiusY="150" />
    </CombinedGeometry.Geometry1>
    <CombinedGeometry.Geometry2>
      <EllipseGeometry Center="400,200" RadiusX="150" RadiusY="150" />
    </CombinedGeometry.Geometry2>
  </CombinedGeometry>

</Path.Data>

<Path.Fill>
  <VisualBrush>
    <VisualBrush.Visual>
      <TextBlock Text="Hello World" FontWeight="Bold" Padding="10">
        <TextBlock.Foreground>
          <LinearGradientBrush>
            <LinearGradientBrush.GradientStops>
              <GradientStop Offset="0" Color="Yellow" />

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

            </LinearGradientBrush.GradientStops>
          </LinearGradientBrush>
        </TextBlock.Foreground>
        <TextBlock.Background>
          <VisualBrush>
            <VisualBrush.Visual>
                  <MediaElement Source="C:\Microsoft Hands-On-Lab\Creating
                  Rich 2D and 3D Content with
                  WPF\Source\apollo11_launchclip04.mpg=" />
            </VisualBrush.Visual>
          </VisualBrush>
        </TextBlock.Background>
      </TextBlock>
    </VisualBrush.Visual>
  </VisualBrush>
</Path.Fill>

<Path.Stroke>
  <LinearGradientBrush>
    <LinearGradientBrush.GradientStops>
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="1" Color="White" />
    </LinearGradientBrush.GradientStops>
  </LinearGradientBrush>
</Path.Stroke>

<Path.Triggers>
  <EventTrigger RoutedEvent="Path.Loaded">
    <EventTrigger.Actions>
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation To="20" Duration="0:0:10"
            Storyboard.TargetName="MyPath"
            Storyboard.TargetProperty="StrokeDashOffset"
            RepeatBehavior="Forever" />
        </Storyboard>
```

```
            </BeginStoryboard>
            <BeginStoryboard>
              <Storyboard>
                <PointAnimation From="150,100" To="650,200" Duration="0:0:3"
                  Storyboard.TargetName="MyEllipseGeometry"
                  Storyboard.TargetProperty="Center"
                  RepeatBehavior="Forever" AutoReverse="True"/>
              </Storyboard>
            </BeginStoryboard>
          </EventTrigger.Actions>
        </EventTrigger>
      </Path.Triggers>
    </Path>
  </Canvas>

</Window>
```

演習 3 – 3-D コンテンツを作成する

この演習では、3-D シーンを作成し、**Viewport3D** を使用してこれを表示することによって、3-D グラフィックスを作成します。

Viewport3D は、3-D シーンを含む 2-D の **FrameworkElement** です。**Viewport3D** では従来の 3-D API と同様に、モデル、ライト、およびカメラを使用して、3-D シーンを指定します。従来の 3-D API と異なる点は、WPF 3-D がプラットフォームに完全に統合されていることです。2-D シーンに 3-D コンテンツを配置できるだけでなく、3-D シーンに 2-D コンテンツを配置することもできます (**DrawingBrush** または **VisualBrush** を使用します)。

作業 1 – 3D シーンを作成する

1. Window1.xaml のコンテンツを消去します。以下のコードを Window1.xaml にコピーして、基本 3-D シーンを作成します。このコードでは、1 つのモデル (2 次元の平面)、2 つのライト、および **PerspectiveCamera** を含む 3-D シーンを指定します。カメラでは視点の場所を定義し、シーンがどのように 2-D に変換されるか (投影の種類) を指定します。ここで使用するカメラの **PerspectiveCamera** では、透視投影を使用します。他にも正投影を使用する **OrthographicCamera**、任意の **ViewMatrix** と **ProjectionMatrix** 操作を可能にする **MatrixCamera** もあります。

```
<Window x:Class="RichContent.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichContent"
    >

<Viewport3D>

    <Viewport3D.Camera>
      <PerspectiveCamera
       FarPlaneDistance="20"
       LookDirection="0,-0.65,-1"
       UpDirection="0,1,0"
       NearPlaneDistance="1"
       Position="0,2,3"
       FieldOfView="40" />
    </Viewport3D.Camera>
```

```
    <Viewport3D.Children>
      <ModelVisual3D>
        <ModelVisual3D.Content>
          <Model3DGroup >
            <Model3DGroup.Children>
              <AmbientLight Color="White" />
              <GeometryModel3D>

                <GeometryModel3D.Geometry>
                  <MeshGeometry3D
                   TriangleIndices="0,1,2   2,3,0   6,5,4   4,7,6"
                   TextureCoordinates="1,1  1,0   0,0   0,1   0,1  0,0   1,0
1,1"
                   Positions="-0.5,-0.5,0   -0.5,0.5,0   0.5,0.5,0    0.5,-0.5,0
-0.5,-0.5,0   -0.5,0.5,0   0.5,0.5,0    0.5,-0.5,0 "/>
                </GeometryModel3D.Geometry>

                <GeometryModel3D.Transform>
                  <RotateTransform3D>
                    <RotateTransform3D.Rotation>
                      <AxisAngleRotation3D x:Name="MyRotation3D" Angle="45"
Axis="0 1 0"/>
                    </RotateTransform3D.Rotation>
                  </RotateTransform3D>
                </GeometryModel3D.Transform>


                <GeometryModel3D.Material>
                  <DiffuseMaterial>
                    <DiffuseMaterial.Brush>
                      <VisualBrush>
                        <VisualBrush.Visual>
                          <TextBlock Text="Hello" />
                        </VisualBrush.Visual>
                      </VisualBrush>
                    </DiffuseMaterial.Brush>
                  </DiffuseMaterial>
                </GeometryModel3D.Material>

              </GeometryModel3D>
            </Model3DGroup.Children>
          </Model3DGroup>
        </ModelVisual3D.Content>
      </ModelVisual3D>
    </Viewport3D.Children>
    <Viewport3D.Triggers>
      <EventTrigger RoutedEvent="Viewport3D.Loaded">
        <EventTrigger.Actions>
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation From="0" To="360" Duration="0:0:3"
              Storyboard.TargetName="MyRotation3D"
              Storyboard.TargetProperty="Angle" RepeatBehavior="Forever" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger.Actions>
      </EventTrigger>
    </Viewport3D.Triggers>
  </Viewport3D>
</Window>
```

図 10: 平面にマップされたテキスト

1. テキストが描画されている位置で **Model3D** を回転するには、**Rotation3D** を使用する **RotateTransform3D** を作成して、Y 軸を中心に回転するコードを記述します。次の変換を使用して、**GeometryModel3D** の **Transform** プロパティを設定します。**Rotation3D** に名前を付けると、次の手順でこのオブジェクトがアニメーション化されます。

```
        …
        </GeometryModel3D.Geometry>

        <GeometryModel3D.Transform>
          <RotateTransform3D>
                <RotateTransform3D.Rotation>
                  <AxisAngleRotation3D
                   x:Name="MyRotation3D" Angle="45" Axis="0 1 0"/>
                </RotateTransform3D.Rotation>
          </RotateTransform3D>
        </GeometryModel3D.Transform>

        <GeometryModel3D.Material>
        …
```

他の種類の 3D 変換には、**ScaleTransform3D** と **TranslateTransform3D** があります。

2. 前の手順で作成した **Rotation3D** をアニメーションにすることで、平面が回転します。

```
  …
</Viewport3D.Models>

<Viewport3D.Triggers>
  <EventTrigger RoutedEvent="Viewport3D.Loaded">
    <EventTrigger.Actions>
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation From="0" To="360" Duration="0:0:3"
```

```
                Storyboard.TargetName="MyRotation3D"
                Storyboard.TargetProperty="Angle" RepeatBehavior="Forever" />
              </Storyboard>
            </BeginStoryboard>
          </EventTrigger.Actions>
        </EventTrigger>
      </Viewport3D.Triggers>

  </Viewport3D>

</Window>
```

図 11: アニメーション化された平面

作業 2 – 立方体を作成する

1. 前の作業で作成した **MeshGeometry3D** を、立方体を定義する **MeshGeometry3D** に置き換えます。

```
<GeometryModel3D.Geometry>

  <MeshGeometry3D
    TriangleIndices="0,1,2 3,4,5 6,7,8 9,10,11 12,13,14 15,16,17 18,19,20
21,22,23 24,25,26 27,28,29 30,31,32 33,34,35 "
    Normals="0,0,-1 0,0,-1 0,0,-1 0,0,-1 0,0,-1 0,0,-1 0,0,-1 0,0,-1 0,0,1 0,0,1 0,0,1 0,0,1
0,0,1 0,0,1 0,-1,0 0,-1,0 0,-1,0 0,-1,0 0,-1,0 0,-1,0 0,-1,0 0,-1,0 1,0,0 1,0,0 1,0,0 1,0,0
1,0,0 1,0,0 0,1,0 0,1,0 0,1,0 0,1,0 0,1,0 0,1,0 0,1,0 -1,0,0 -1,0,0 -1,0,0 -1,0,0 -
1,0,0 -1,0,0 "
    TextureCoordinates="1,1 1,0 0,0 0,0 0,1 1,1 0,1 1,1 1,0 1,0 0,0 0,1 0,1 1,1
1,0 1,0 0,0 0,1 1,1 1,0 0,0 0,0 0,1 1,1 1,0 0,0 0,0 0,1 0,1 1,1 1,0 0,0 0,1 1,1 1,1
1,0 0,0 "
    Positions="-0.5,-0.5,-0.5 -0.5,0.5,-0.5 0.5,0.5,-0.5 0.5,0.5,-0.5 0.5,-0.5,-
0.5 -0.5,-0.5,-0.5 -0.5,-0.5,0.5 0.5,-0.5,0.5 0.5,0.5,0.5 0.5,0.5,0.5 -
0.5,0.5,0.5 -0.5,-0.5,0.5 -0.5,-0.5,-0.5 0.5,-0.5,-0.5 0.5,-0.5,0.5 -0.5,-0.5,0.5
-0.5,-0.5,0.5 -0.5,-0.5,-0.5 0.5,-0.5,-0.5 0.5,0.5,-0.5 0.5,0.5,0.5 0.5,0.5,0.5
0.5,-0.5,0.5 0.5,-0.5,-0.5 0.5,0.5,-0.5 -0.5,0.5,-0.5 -0.5,0.5,0.5 -0.5,0.5,0.5
```

```
0.5,0.5,0.5 0.5,0.5,-0.5 -0.5,0.5,-0.5 -0.5,-0.5,-0.5 -0.5,-0.5,0.5 -0.5,-0.5,0.5
-0.5,0.5,0.5 -0.5,0.5,-0.5 "/>

</GeometryModel3D.Geometry>
```

テキストが貼り付けられた、回転する立方体が作成されます。カメラの **Position** プロパティの設定を変更してみてください。

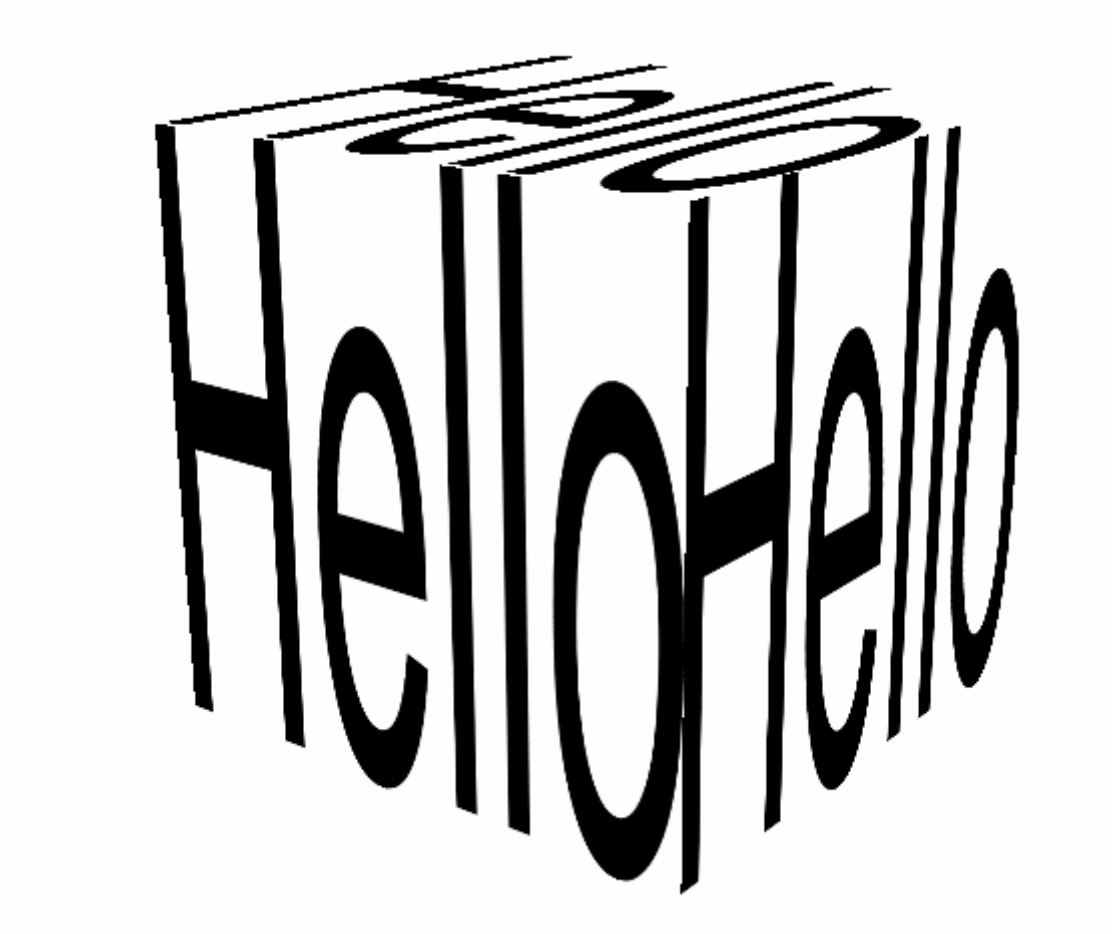


図 12: 立方体にマップされたテキスト

作業 3 – 立方体の表面に他のコンテンツをマップする

1. 立方体の表面を変更して、ビデオを表示します。**VisualBrush** のコンテンツを変更することで、この作業を行います。具体的には、**TextBlock** を…

```
<DiffuseMaterial>
  <DiffuseMaterial.Brush>
    <VisualBrush>
      <VisualBrush.Visual>
        <TextBlock Text="Hello" />
      </VisualBrush.Visual>
    </VisualBrush>
  </DiffuseMaterial.Brush>
</DiffuseMaterial>
```

…**MediaElement** に変更します。

```
<DiffuseMaterial>
  <DiffuseMaterial.Brush>
    <VisualBrush>
      <VisualBrush.Visual>
        <MediaElement Source = "C:\Microsoft Hands-On-Lab\Creating Rich 2D and 3D
Content with WPF\Source\apollo11_launchclip04.mpg" />
      </VisualBrush.Visual>
    </VisualBrush>
  </DiffuseMaterial.Brush>
</DiffuseMaterial>
```

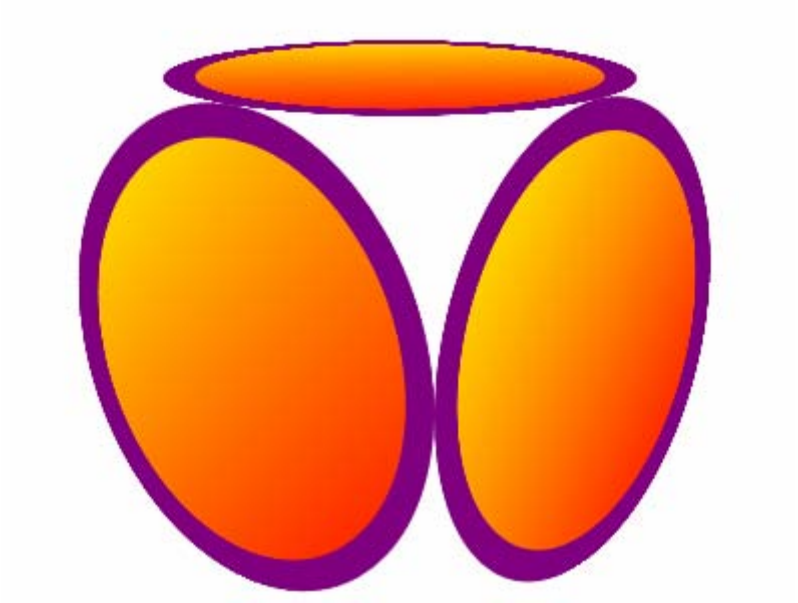
2. **VisualBrush** のコンテンツを再び変更します。今度は **Ellipse** に置き換えます。

```
<DiffuseMaterial>
  <DiffuseMaterial.Brush>
    <VisualBrush>
      <VisualBrush.Visual>

        <Ellipse Width="300" Height="300"
          Stroke="Purple" StrokeThickness="20">
          <Ellipse.Fill>
            <LinearGradientBrush>
              <LinearGradientBrush.GradientStops>
                <GradientStop Offset="0" Color="Yellow" />
                <GradientStop Offset="1" Color="Red" />
              </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
```

```
          </Ellipse.Fill>
        </Ellipse>

      </VisualBrush.Visual>
    </VisualBrush>
  </DiffuseMaterial.Brush>
</DiffuseMaterial>
```



作業 4 – 立方体にアニメーションにした 2-D コンテンツをマップする

1. 演習 2 で作成した **Path** を、立方体の表面にマップします。この作業を行うには、前の作業の **Ellipse** を削除し、演習 2 のパスに置き換えます。

```
        <GeometryModel3D.Material>
          <DiffuseMaterial>
            <DiffuseMaterial.Brush>
              <VisualBrush>
                <VisualBrush.Visual>

                  <Path Name="MyPath"
                    Width="800" Height="800"
                    StrokeThickness="20" StrokeDashArray="0.5,2"
                    StrokeDashCap="Round">

                    <Path.Data>
                      <CombinedGeometry GeometryCombineMode="Union">
                        <CombinedGeometry.Geometry1>
                          <EllipseGeometry x:Name="MyEllipseGeometry"
                            Center="200 200" RadiusX="150" RadiusY="150" />
                        </CombinedGeometry.Geometry1>
                        <CombinedGeometry.Geometry2>
                          <EllipseGeometry Center="400 200" RadiusX="150"
RadiusY="150" />
                        </CombinedGeometry.Geometry2>
                      </CombinedGeometry>
                    </Path.Data>
```

```
                        <Path.Fill>
                          <VisualBrush>
                            <VisualBrush.Visual>
                              <TextBlock Text="Hello World" FontWeight="Bold"
Padding="10">
                                <TextBlock.Foreground>
                                  <LinearGradientBrush>
                                    <LinearGradientBrush.GradientStops>
                                      <GradientStop Offset="0" Color="Yellow" />
                                      <GradientStop Offset="1" Color="Red" />
                                    </LinearGradientBrush.GradientStops>
                                  </LinearGradientBrush>
                                </TextBlock.Foreground>
                                <TextBlock.Background>
                                  <VisualBrush>

                                    <VisualBrush.Visual>
                                      <MediaElement>
                                        <MediaElement.Source>
                                          <MediaTimeline Source="C:\Microsoft
Hands-On-Lab\Creating Rich 2D and 3D Content with
WPF\Source\apollo11_launchclip04.mpg" RepeatBehavior="Forever"/>
                                        </MediaElement.Source>
                                       </MediaElement>

                                    </VisualBrush.Visual>
                                  </VisualBrush>
                                </TextBlock.Background>
                              </TextBlock>
                            </VisualBrush.Visual>
                          </VisualBrush>
                        </Path.Fill>

                        <Path.Stroke>
                          <LinearGradientBrush>
                            <LinearGradientBrush.GradientStops>
                              <GradientStop Offset="0" Color="Black" />
                              <GradientStop Offset="1" Color="White" />
                            </LinearGradientBrush.GradientStops>
                          </LinearGradientBrush>
                        </Path.Stroke>

                        <Path.Triggers>
                          <EventTrigger RoutedEvent="Path.Loaded">
                            <EventTrigger.Actions>
                              <BeginStoryboard>
                                <Storyboard>
                                  <DoubleAnimation To="20" Duration="0:0:10"
                                    Storyboard.TargetName="MyPath"
                                    Storyboard.TargetProperty="StrokeDashOffset"
                                    RepeatBehavior="Forever" />
                                </Storyboard>
                              </BeginStoryboard>
                              <BeginStoryboard>
                                <Storyboard>
                                  <PointAnimation From="150,100" To="650,200"
Duration="0:0:3"
                                    Storyboard.TargetName="MyEllipseGeometry"
                                    Storyboard.TargetProperty="Center"
```

```
                                    RepeatBehavior="Forever" AutoReverse="True"/>
                                </Storyboard>
                            </BeginStoryboard>
                        </EventTrigger.Actions>
                    </EventTrigger>
                </Path.Triggers>
            </Path>

                    </VisualBrush.Visual>
                </VisualBrush>
            </DiffuseMaterial.Brush>
        </DiffuseMaterial>

    </GeometryModel3D.Material>

</GeometryModel3D>
```

図 13: 立方体の表面にマップされた演習 2 のパス

2. アニメーションを再び機能させるには、**DashOffset** アニメーションと **EllipseGeometry** アニメーションを、パスの **EventTrigger** から Viewport3D の **EventTrigger** に移動します。パスの Path.Trigger タグとその内部のコードをすべて削除します。

```
<Viewport3D.Triggers>
  <EventTrigger RoutedEvent="Viewport3D.Loaded">
```

```
      <EventTrigger.Actions>
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation From="0" To="360" Duration="0:0:3"
              Storyboard.TargetName="MyRotation3D"
              Storyboard.TargetProperty="Angle" RepeatBehavior="Forever" />
          </Storyboard>
        </BeginStoryboard>

        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation To="20" Duration="0:0:10"
              Storyboard.TargetName="MyPath"
              Storyboard.TargetProperty="StrokeDashOffset"
              RepeatBehavior="Forever" />
          </Storyboard>
        </BeginStoryboard>
        <BeginStoryboard>
          <Storyboard>
            <PointAnimation From="150,100" To="650,200" Duration="0:0:3"
              Storyboard.TargetName="MyEllipseGeometry"
              Storyboard.TargetProperty="Center"
              RepeatBehavior="Forever" AutoReverse="True"/>
          </Storyboard>
        </BeginStoryboard>

      </EventTrigger.Actions>
    </EventTrigger>
  </Viewport3D.Triggers>
```

実習のまとめ

この実習では、以下の演習を行いました。

- 2-D シーンを作成しました
- 2-D シーンをアニメーションにし、再生するビデオを追加しました
- アニメーションにした 3-D 立方体の表面に、シーン全体をマップしました

この実習全体は XAML で記述しているので、すべてを XAMLPad を使用して実行できます。XAMLPad は、各キーストロークで xaml を再解析することによって、xaml の編集時にリアルタイムに更新できる XAML エディタおよびビューアです。XAMLPad は、WPF での作業をテストするのに適しています。

[スタート (Start)] ボタンをクリックし、[すべてのプログラム (All Programs)] をポイントして、[Microsoft Windows SDK] を起動します。[ツール (Tools)] メニューの [XAMLPad] をクリックします。XAMLPad は、終了するときに自動的に状態を保存するので、XAMLPad のテキスト入力ボックスに、既に XAML が表示されていることがあります。表示されている XAML はすべて削除してください。