Microsoft

Windows 8®

Programowanie aplikacji z wykorzystaniem C# i XAML

Wydanie VI

Professional

Helion

Charles Petzold

Tytuł oryginału: Programming Windows: Writing Windows 8 Apps With C# and XAML

Tłumaczenie: Paweł Gonera

ISBN: 978-83-246-7383-4

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/win8pa.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/win8pa
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 18.

Sensory i GPS

W ostatnim czasie komputery zostały wyposażone w nowe „zmysły". To nie jest szkic nowego filmu! Wiele komputerów — a w szczególności tabletów i innych urządzeń mobilnych — posiada sensory, które pozwalają maszynie zorientować się w przestrzeni trójwymiarowej, w położeniu na powierzchni Ziemi, sprawdzić ilość światła, a nawet szybkość, z jaką użytkownik obraca w rękach komputer.

Te elementy sprzętowe, nazywane ogólnie *sensorami*, i interfejsy programowe pozwalające na korzystanie z nich znajdują się w większości w przestrzeni nazw `Windows.Devices.Sensors`, natomiast klasy pomagające programowi w określeniu położenia geograficznego znajdują się w przestrzeni nazw `Windows.Devices.Geolocation`. Sprzęt realizujący to drugie zadanie jest często nazywany nieformalnie skrótem GPS (ponieważ tego typu system satelitarny nosi nazwę Global Positioning System), ale często komputer może określić położenie geograficzne z użyciem połączenia sieciowego.

W tym rozdziale skupimy się na danych dostępnych poprzez klasy `SimpleOrientationSensor`, `Accelerometer`, `Compass`, `Inclinometer`, `OrientationSensor` oraz `Geolocator`, ale muszę pominąć rzadziej używane klasy `LightSensor` i `Gyrometer`, które pozwalają na pomiar ilości światła oraz szybkości kątowej komputera.

Aby w pełni wykorzystać informacje z tego rozdziału, powinieneś wziąć do ręki komputer z uruchomionymi programami i poruszyć nim, a nawet trzymać go nad głową. Jeżeli Twój komputer, na którym tworzysz aplikacje Windows 8, jest przywiązany do biurka tak jak mój, możesz postarać się o tablet, taki jak Microsoft Surface, i zainstalować na nim programy w sposób pokazany przez Tima Heuera na jego blogu, dostępnym pod adresem *http://timheuer.com/blog/archive/2012/10/26/remote-debugging-windows-store-apps-on-surface-arm-devices.aspx*.

Część przykładowych programów z tego rozdziału powstało na podstawie artykułów, jakie napisałem na temat sensorów w Windows Phone 7.5, opublikowanych od czerwca do grudnia 2012 r. w „MSDN Magazine".

Orientacja

Najprostszą klasą sensorów, jaką się zajmiemy, jest `SimpleOrientationSensor`, która pozwala w przybliżony sposób zorientować się, jakie jest położenie komputera w przestrzeni trójwymiarowej, ale bez większych szczegółów. Aby utworzyć obiekt `SimpleOrientationSensor`, korzystamy z metody statycznej:

```
SimpleOrientationSensor simpleOrientationSensor = SimpleOrientationSensor.GetDefault();
```

Operację tę wykonujemy w aplikacji tylko raz, więc kod ten powinien pojawić się jako definicja pola, co pozwoli na dostęp do tego obiektu dla całej klasy. Jeżeli metoda `SimpleOrientationSensor.GetDefault` zwróci `null`, komputer nie posiada urządzenia pozwalającego na odczyt orientacji.

W dowolnym momencie z obiektu `SimpleOrientationSensor` możemy odczytać wartość oznaczającą bieżącą orientację:

```
SimpleOrientation simpleOrientation = simpleOrientationSensor.GetCurrentOrientation();
```

`SimpleOrientation` to typ wyliczeniowy z sześcioma składowymi:

- `NotRotated`,
- `Rotated90DegreesCounterclockwise`,
- `Rotated180DegreesCounterclockwise`,
- `Rotated270DegreesCounterclockwise`,
- `Faceup`,
- `Facedown`.

Ograniczenie ilości danych do tych sześciu wartości tłumaczy, dlaczego w nazwie klasy `SimpleOrientationSensor` znalazła się fraza *simple* — prosta.

Możemy również być powiadomieni o zmianie orientacji za pomocą zdarzenia. Ustawienie metody obsługi zdarzenia `OrientationChanged` jest realizowane w następujący sposób:

```
simpleOrientationSensor.OrientationChanged += OnSimpleOrientationChanged;
```

Zdarzenie to jest generowane w momencie zmiany orientacji, co nie dzieje się, gdy komputer jest względnie nieruchomy. Jeżeli potrzebujesz wartości początkowej, to oprócz ustawienia metody obsługi zdarzenia wywołaj metodę `GetCurrentOrientation`.

Metoda obsługi zdarzenia działa we własnym wątku, więc interakcja z wątkiem interfejsu użytkownika musi zachodzić z użyciem obiektu `CoreDispatcher`:

```
async void OnSimpleOrientationChanged(SimpleOrientationSensor sender,
                  SimpleOrientationSensorOrientationChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
     ...
    });
}
```

Argument zdarzenia mający typ o bardzo długiej nazwie posiada właściwość `Orien`
`↪tation` typu wyliczeniowego `SimpleOrientation` oraz właściwość `Timestamp` typu `Date`
`↪TimeOffset`.

Możesz tu zadać pytania: „Czy nie korzystałem już z tych danych orientacji? Czy nie są one dostępne w przestrzeni nazw `Windows.Graphics.Display`? Czy nie korzystałem z klasy `DisplayProperties` i jej właściwości statycznych `NativeOrientation` oraz `Current`
`↪Orientation`, jak również zdarzenia `OrientationChanged` do odczytania danych o orientacji?”. Na pewno pamiętasz, że te dwie właściwości statyczne zwracają wartości typu wyliczeniowego `DisplayOrientations`:

- `Landscape`,
- `Portrait`,
- `LandscapeFlipped`,
- `PortraitFlipped`.

Klasy `SimpleOrientationSensor` i `DisplayProperties` są oczywiście ze sobą związane, ale trzeba wiedzieć, w jaki sposób. Klasa `SimpleOrientationSensor` pozwala sprawdzić, jak komputer jest zorientowany w przestrzeni. Właściwość `DisplayProperties.Current` `↪Orientation` pokazuje, jak Windows dostosował się do orientacji komputera przez zmianę orientacji okna programu. Inaczej mówiąc, `SimpleOrientationSensor` raportuje orientację sprzętu, natomiast `DisplayProperties.CurrentOrientation` raportuje orientację oprogramowania, które zareagowało na zmianę w położeniu sprzętu.

Projekt *OrientationAndOrientation* ma za zadanie pokazać różnice pomiędzy tymi orientacjami. W pliku XAML zdefiniowanych jest kilka elementów `TextBlock` dla etykiet oraz do wyświetlania informacji:

Listing 18.1. Projekt: OrientationAndOrientation | Plik: MainPage.xaml (fragment)

```
<Page... FontSize="18">
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid HorizontalAlignment="Center"
              VerticalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>

            <TextBlock Text="SimpleOrientationSensor: "
                       Grid.Row="0"
                       Grid.Column="0" />

            <TextBlock Name="orientationSensorTextBlock"
                       Grid.Row="0"
                       Grid.Column="1"
                       TextAlignment="Right" />

            <TextBlock Text="DisplayProperties.CurrentOrientation: "
                       Grid.Row="1"
                       Grid.Column="0" />

            <TextBlock Name="displayOrientationTextBlock"
                       Grid.Row="1"
                       Grid.Column="1"
                       TextAlignment="Right" />
        </Grid>
    </Grid>
</Page>
```

W pliku kodu ukrytego zdefiniowane są dwie metody mające za zadanie ustawić dwa elementy `TextBlock` w drugiej kolumnie elementu `Grid`. Te dwie metody są wywoływane zarówno w konstruktorze, aby ustawić wartości początkowe, jak i z dwóch metod obsługi zdarzeń.

Listing 18.2. Projekt: OrientationAndOrientation | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    SimpleOrientationSensor simpleOrientationSensor =
    ↪SimpleOrientationSensor.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();

        // Inicjalizacja SimpleOrientationSensor
        if (simpleOrientationSensor != null)
        {
            SetOrientationSensorText(simpleOrientationSensor.GetCurrentOrientation());
            simpleOrientationSensor.OrientationChanged += OnSimpleOrientationChanged;
        }

        // Inicjalizacja DisplayProperties
        SetDisplayOrientationText(DisplayProperties.CurrentOrientation);
        DisplayProperties.OrientationChanged += OnDisplayPropertiesOrientationChanged;
    }

    // Metoda obsługująca SimpleOrientationSensor
    async void OnSimpleOrientationChanged(SimpleOrientationSensor sender,
                                          SimpleOrientationSensorOrientation
                                          ↪ChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                SetOrientationSensorText(args.Orientation);
            });
    }

    void SetOrientationSensorText(SimpleOrientation simpleOrientation)
    {
        orientationSensorTextBlock.Text = simpleOrientation.ToString();
    }

    // Metoda obsługująca DisplayProperties
    void OnDisplayPropertiesOrientationChanged(object sender)
    {
        SetDisplayOrientationText(DisplayProperties.CurrentOrientation);
    }

    void SetDisplayOrientationText(DisplayOrientations displayOrientation)
    {
        displayOrientationTextBlock.Text = displayOrientation.ToString();
    }
}
```

Zwróć uwagę, że obiekt `SimpleOrientationSensor` jest tworzony jako pole, ale konstruktor sprawdza przed dostępem do niego, czy zawiera wartość różną od `null`.

Jeżeli uruchomisz ten program na tablecie mającym standardową orientację poziomą — czyli właściwość `DisplayProperties.NativeOrientation` zawiera wartość `Display`
`↪Orientations.Landscape` — i jeżeli nie zrobiłeś nic, aby uniemożliwić Windows zmianę orientacji (na przykład umieszczając tablet w stacji dokującej), to w czasie obracania tabletu w kierunku zgodnym z ruchem wskazówek zegara *zazwyczaj* można zauważyć następującą korelację pomiędzy dwoma wskaźnikami orientacji:

SimpleOrientationSensor	DisplayProperties.CurrentOrientation
`NotRotated`	`Landscape`
`Rotated270DegreesCounterClockwise`	`Portrait`
`Rotated180DegreesCounterClockwise`	`LandscapeFlipped`
`Rotated90DegreesCounterClockwise`	`PortraitFlipped`

Obiekt `SimpleOrientationSensor` raportuje również wartości `Faceup` i `Facedown`, które nie mają odpowiedników w typie `DisplayOrientations`.

Choć powyższa tabela zwykle zgadza się dla tabletów z natywną orientacją poziomą, to jednak urządzenia mające standardowo orientację pionową wykazują następującą korelację:

SimpleOrientationSensor	DisplayProperties.CurrentOrientation
`NotRotated`	`Portrait`
`Rotated270DegreesCounterClockwise`	`LandscapeFlipped`
`Rotated180DegreesCounterClockwise`	`PortraitFlipped`
`Rotated90DegreesCounterClockwise`	`Landscape`

Co więcej, aplikacja może zażądać od Windows, aby nie wykonywał zmian kompensujących orientację komputera — czy to za pomocą ustawienia w pliku *Package.appx manifest*, czy to programowo, przez ustawienie właściwości `DisplayProperties.Auto`
`↪RotationPreferences`. W takim przypadku wartość `DisplayProperties.CurrentOrien`
`↪tation` nie zmienia się w czasie działania aplikacji. Niektóre tablety mają przełącznik sprzętowy pozwalający zablokować obracanie ekranu przez Windows. W takim przypadku możesz nawet zobaczyć coś takiego:

SimpleOrientationSensor	DisplayProperties.CurrentOrientation
`NotRotated`	`PortraitFlipped`
`Rotated270DegreesCounterClockwise`	`PortraitFlipped`
`Rotated180DegreesCounterClockwise`	`PortraitFlipped`
`Rotated90DegreesCounterClockwise`	`PortraitFlipped`

Jeżeli chcesz, możesz nawet samodzielnie wykonać kompensację orientacji. Możesz poinstruować Windows, aby nie były wykonywane zmiany orientacji, a następnie użyć `SimpleOrientationSensor` do sprawdzenia, jak jest zorientowany komputer. Jednak pamiętaj, że ustawienie z pliku *Package.appxmanifest* oraz `DisplayProperties.AutoRo`
`↪tationPreferences` jedynie informuje Windows o Twoich preferencjach, więc gdy Windows postąpi niezgodnie z tymi wskazówkami, mogą być konieczne dalsze korekty.

Chyba najbezpieczniejszym podejściem do wyłączenia automatycznego obrotu jest ustawienie właściwości `DisplayProperties.AutoRotationPreferences` na `DisplayPro`
`↪perties.NativeOrientation`, z czego będziemy korzystać w dalszej części rozdziału.

Przyspieszenie, siła, grawitacja i wektory

Klasa `SimpleOrientationSensor` bez wątpienia wykorzystuje urządzenie sprzętowe nazywane *przyspieszeniomierzem* albo *akcelerometrem*. Przyspieszeniomierz jest urządzeniem mierzącym przyspieszenie, na początek może się wydawać, że znajomość przyspieszenia nie jest zbyt przydatna. Jednak wiemy z fizyki — a dokładniej z drugiego prawa dynamiki Newtona — że:

$$F = ma$$

Siła jest równa iloczynowi masy i przyspieszenia, a jedną z sił, od której trudno uciec, jest siła grawitacji. Przez większość czasu przyspieszeniomierz komputera mierzy grawitację i odpowiada na proste pytanie, gdzie jest dół.

Bardziej bezpośredni dostęp do przyspieszeniomierza mamy poprzez klasę `Accele`
`↪rometer`. Aby utworzyć obiekt klasy `Accelerometer`, należy użyć metody statycznej o tej samej nazwie co w `SimpleOrientationSensor`:

```
Accelerometer accelerometer = Accelerometer.GetDefault();
```

Jeżeli metoda `Accelerometer.GetDefault` zwróci `null`, komputer nie posiada przyspieszeniomierza albo Windows 8 nic nie wie na jego temat. Jeżeli Twoja aplikacja nie może działać bez przyspieszeniomierza, powinieneś poinformować użytkownika o jego braku.

W dowolnym momencie możesz odczytać bieżącą wartość przyspieszenia:

```
AccelerometerReading accelerometerReading = accelerometer.GetCurrentReading();
```

Podobna metoda w klasie `SimpleOrientationSensor` nosi nazwę `GetCurrentOrientation`.

Dobrym pomysłem jest sprawdzanie, czy wartością zwracaną przez `GetCurrentReading` jest `null`. W klasie `AccelerometerReading` są zdefiniowane cztery właściwości:

- `AccelerationX` typu `double`,
- `AccelerationY` typu `double`,
- `AccelerationZ` typu `double`,
- `Timestamp` typu `DateTimeOffset`.

Trzy wartości `double` tworzą wektor trójwymiarowy wskazujący od urządzenia w kierunku ziemi. Więcej na jego temat wkrótce.

Do obiektu `Accelerometer` można również dołączyć metodę obsługi zdarzenia:

```
accelerometer.ReadingChanged += OnAccelerometerReadingChanged;
```

Podobne zdarzenie w klasie `SimpleOrientationSensor` nosi nazwę `OrientationChanged`. Podobnie jak w przypadku `OrientationChanged`, metoda obsługująca `ReadingChanged` działa w osobnym wątku, więc powinna być obsługiwana w następujący sposób:

```
async void OnAccelerometerReadingChanged(Accelerometer sender,
AccelerometerReadingChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
       ...
    });
}
```

Klasa `AccelerometerReadingChangedEventArgs` ma zdefiniowaną właściwość o nazwie `Reading` typu `AccelerometerReading`, takiego samego jak obiekt zwracany przez `GetCur`
`↪rentReading`.

Jak często będzie wywoływana metoda obsługi zdarzenia `ReadingChanged`? Jeżeli komputer jest nieruchomy, być może wcale! Z tego powodu, jeżeli potrzebujesz początkowych odczytów przyspieszenia, powinieneś na początek wywołać metodę `GetCurrent`
`↪Reading`.

Jeżeli komputer jest przenoszony lub zmienia orientację w przestrzeni, metoda `Reading`
`↪Changed` jest wywoływana przy zmianie wartości (przy określonych kryteriach), ale pomiędzy kolejnymi wywołaniami następuje przerwa o długości nie mniejszej niż liczba milisekund odczytana z właściwości `ReportInterval` obiektu `Accelerometer`. Z właściwości tej odczytałem wartość 112, czyli metoda `ReadingChanged` jest wywoływana nie częściej niż dziewięć razy na sekundę.

Właściwości `ReportInterval` można przypisać inną wartość, ale nie niższą od wartości z właściwości `MinimumReportInterval`, która wynosi 16 milisekund, czyli około 60 razy na sekundę. Ustawienie `ReportInterval` na `MinimumReportInterval` pozwala uzyskać maksymalną ilość danych; ustawienie `ReportInterval` na zero powoduje powrót do ustawień domyślnych.

Pozostałe klasy czujników w przestrzeni nazw `Windows.Devices.Sensors` mają ten sam interfejs programowy co `Accelerometer`. Mają one następujące składniki:

- statyczną metodę `GetDefault`,
- metodę instancyjną `GetCurrentReading`,
- właściwość `ReportInterval`,
- właściwość tylko do odczytu `MinimumReportInterval`,
- zdarzenie `ReadingChanged`.

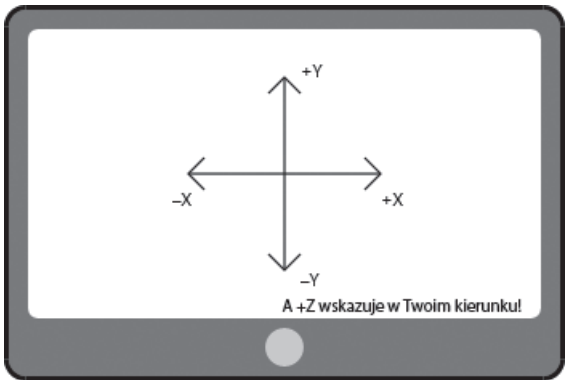
Jedynie klasa `SimpleOrientationSensor` różni się od pozostałych.

Jeżeli komputer jest nieruchomy, właściwości `AccelerationX`, `AccelerationY` oraz `AccelerationZ` klasy `AccelerometerReading` definiują wektor wskazujący w kierunku środka Ziemi. Wektory są zwykle zapisywane pogrubieniem jako współrzędne (**x**, **y**, **z**), dla odróżnienia od punktów (*x*, *y*, *z*). Punkt jest lokalizacją w przestrzeni, natomiast wektor jest kierunkiem i modułem. Oczywiście zachodzi związek pomiędzy wektorem i punktem. Kierunek wektora (**x**, **y**, **z**) jest kierunkiem z punktu (0, 0, 0) do punktu (*x*, *y*, *z*), a modułem wektora jest długość tego odcinka. Jednak sam wektor nie jest odcinkiem i nie posiada lokalizacji.

Moduł wektora może być wyliczony za pomocą trójwymiarowej wersji twierdzenia Pitagorasa:

$$Moduł = \sqrt{x^2 + y^2 + z^2}$$

Każdy wektor trójwymiarowy musi być usytuowany w określonym układzie współrzędnych trójwymiarowych i wektor odczytany z `AccelerometerReading` nie jest wyjątkiem. W przypadku tabletu z natywną orientacją poziomą układ współrzędnych jest następujący:



Zwróć uwagę, że wartości *Y* zwiększają się w górę, czyli odwrotnie niż w przypadku konwencji stosowanych w grafice dwuwymiarowej. Dodatnia oś *Z* „wystaje" z ekranu. Konwencja taka jest często nazywana układem współrzędnych prawej ręki. Jeżeli palec wskazujący prawej ręki wskazuje w kierunku dodatnich *X*, duży palec w kierunku dodatnich *Y*, to kciuk wskazuje w kierunku dodatnich *Z*.

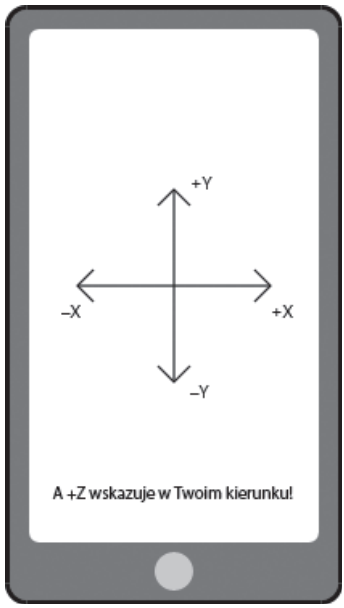
Jeżeli obrócisz palce prawej ręki w kierunku potrzebnym do obrotu dodatniej osi *X* w kierunku dodatniej osi *Y*, kciuk będzie wskazywał w kierunku dodatniej osi *Z*. Działa to dla dowolnych par osi, w kolejności *X*, *Y*, *Z*. Obróć palce prawej dłoni w taki sposób, aby dodatnia oś *Y* przesunęła się w kierunku dodatniej osi *Z*, a kciuk będzie wskazywał w kierunku dodatniej osi *X*. Albo obróć palce prawej dłoni w taki sposób, aby dodatnia oś *Z* przesunęła się w kierunku dodatniej osi *X*, a kciuk będzie wskazywał w kierunku dodatniej osi *Y*.

Zasada prawej dłoni może być używana do określenia kierunku obrotu wokół osi. W przypadku obrotów wokół osi *X* (dla przykładu) wskaż kciukiem prawej dłoni w kierunku dodatniej osi *X*, a palce będą obracać się w kierunku dodatnich kątów obrotu wokół osi.

W przypadku urządzeń z natywną orientacją pionową układ współrzędnych jest taki sam z perspektywy użytkownika (patrz rysunek na następnej stronie).

Choć nie byłem w stanie tego potwierdzić, układ współrzędnych dla zwykłych laptopów bazuje na powierzchni klawiatury, a nie ekranu. Osią *X* jest dłuższy bok klawiatury, osią *Y* krótszy bok klawiatury, a oś *Z* wskazuje ponad klawiaturę.

Ten układ współrzędnych jest stały dla urządzenia i wektor obiektu `Accelerometer` wskazuje w kierunku środka Ziemi, mając wartości określane względem tego układu współrzędnych. Gdy tablet jest na przykład trzymany prosto w swojej natywnej orientacji,



wektor przyspieszenia wskazuje w kierunku *–Y*. Moduł wektora wynosi mniej więcej 1, więc wektor ten można zapisać jako (**0**, **–1**, **0**). Gdy urządzenie leży na płaskiej powierzchni stołu, ekranem w górę, to wektor ma wartość bliską (**0**, **0**, **–1**).

Moduł równy 1 wskazuje, że wektor ten jest mierzony w jednostkach g, czyli przyspieszeniem powodowanym przez siłę grawitacji na powierzchni Ziemi, tj. 9,81 metra na sekundę do kwadratu. Jeżeli zabierzesz tablet na Księżyc, moduł wektora spadnie do 0,17. Jeżeli tablet będzie spadał swobodnie (o ile się odważysz rzucić go) to moduł wektora przyspieszenia zmniejszy się do zera, aż do momentu uderzenia w podłoże.

Przedstawię teraz program *AccelerometerAndSimpleOrientation*, który wyświetla wartości z obiektów `Accelerometer` oraz `SimpleOrientationSensor`. Plik XAML zawiera zestaw elementów `TextBlock` na etykiety i wartości ustawiane za pomocą kodu ukrytego.

Listing 18.3. Projekt: AccelerometerAndSimpleOrientation | Plik: MainPage.xaml (fragment)

```
<Page ... >

    <Page.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="FontSize" Value="24" />
            <Setter Property="Margin" Value="24 12 24 12" />
        </Style>
    </Page.Resources>

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">

        <Grid HorizontalAlignment="Center"
              VerticalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
```

```
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>

        <TextBlock Grid.Row="0" Grid.Column="0" Text="Przyspieszeniomierz X:" />
        <TextBlock Grid.Row="1" Grid.Column="0" Text="Przyspieszeniomierz Y:" />
        <TextBlock Grid.Row="2" Grid.Column="0" Text="Przyspieszeniomierz Z:" />
        <TextBlock Grid.Row="3" Grid.Column="0" Text="Moduł:"
                   Margin="24 24" />
        <TextBlock Grid.Row="4" Grid.Column="0" Text="Orientacja:" />

        <TextBlock Grid.Row="0" Grid.Column="1" Name="accelerometerX"
                   TextAlignment="Right" />
        <TextBlock Grid.Row="1" Grid.Column="1" Name="accelerometerY"
                   TextAlignment="Right"/>
        <TextBlock Grid.Row="2" Grid.Column="1" Name="accelerometerZ"
                   TextAlignment="Right"/>
        <TextBlock Grid.Row="3" Grid.Column="1" Name="magnitude"
                   TextAlignment="Right"
                   VerticalAlignment="Center" />
        <TextBlock Grid.Row="4" Grid.Column="1" Name="simpleOrientation"
                   TextAlignment="Right" />
    </Grid>
</Grid>
</Page>
```

Plik kodu ukrytego posiada nieco więcej kodu sprawdzającego niż poprzednio. Jeżeli nie można utworzyć obiektu `Accelerometer` lub `SimpleOrientationSensor`, informacja o tym jest wyświetlana użytkownikowi. Dodatkowo dobrym pomysłem jest, aby obiekt `Accelerometer` nie działał w czasie, gdy program z niego nie korzysta, ponieważ powoduje to szybsze zużywanie baterii. Aby pokazać, że aplikacja szanuje baterię, program ten podłącza metodę obsługi w `OnNavigateTo` i odłącza ją w `OnNavigateFrom`. Poza tym jego struktura jest zbliżona do poprzedniego programu.

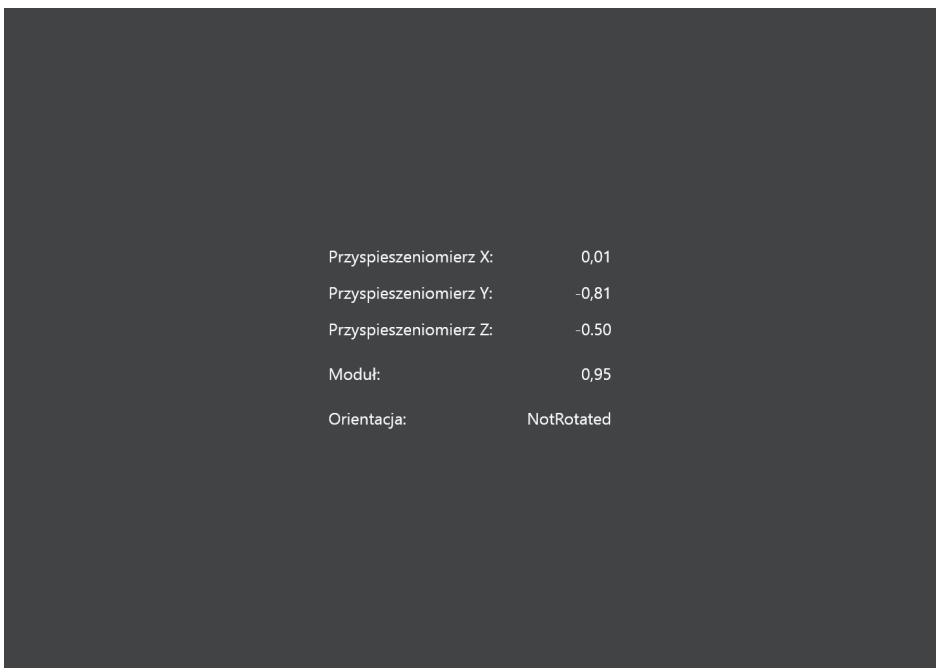
Listing 18.4. Projekt: AccelerometerAndSimpleOrientation | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Accelerometer accelerometer = Accelerometer.GetDefault();
    SimpleOrientationSensor simpleOrientationSensor =
    ↪SimpleOrientationSensor.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        this.Loaded += OnMainPageLoaded;
    }

    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (accelerometer == null)
```

```
            await new MessageDialog("Nie można uruchomić
            ↪przyspieszeniomierza").ShowAsync();

        if (simpleOrientationSensor == null)
            await new MessageDialog("Nie można uruchomić
            ↪SimpleOrientationSensor").ShowAsync();
    }

    // Podłączenie metod obsługi zdarzeń
    protected override void OnNavigatedTo(NavigationEventArgs args)
    {
        if (accelerometer != null)
        {
            SetAccelerometerText(accelerometer.GetCurrentReading());
            accelerometer.ReadingChanged += OnAccelerometerReadingChanged;
        }

        if (simpleOrientationSensor != null)
        {
            SetSimpleOrientationText(simpleOrientationSensor.GetCurrentOrientation());
            simpleOrientationSensor.OrientationChanged += OnSimpleOrientationChanged;
        }
        base.OnNavigatedTo(args);
    }

    // Odłączenie metod obsługi zdarzeń
    protected override void OnNavigatedFrom(NavigationEventArgs args)
    {
        if (accelerometer != null)
            accelerometer.ReadingChanged -= OnAccelerometerReadingChanged;

        if (simpleOrientationSensor != null)
            simpleOrientationSensor.OrientationChanged -= OnSimpleOrientationChanged;

        base.OnNavigatedFrom(args);
    }

    // Metoda obsługi przyspieszeniomierza
    async void OnAccelerometerReadingChanged(Accelerometer sender,
                                             AccelerometerReadingChangedEventArgs
                                             ↪args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                SetAccelerometerText(args.Reading);
            });
    }

    void SetAccelerometerText(AccelerometerReading accelerometerReading)
    {
        if (accelerometerReading == null)
            return;

        accelerometerX.Text = accelerometerReading.AccelerationX.ToString("F2");
        accelerometerY.Text = accelerometerReading.AccelerationY.ToString("F2");
        accelerometerZ.Text = accelerometerReading.AccelerationZ.ToString("F2");
        magnitude.Text =
```

```
            Math.Sqrt(Math.Pow(accelerometerReading.AccelerationX, 2) +
                        Math.Pow(accelerometerReading.AccelerationY, 2) +
                        Math.Pow(accelerometerReading.AccelerationZ,
                        ↪2)).ToString("F2");
    }

    // Metoda obsługi SimpleOrientationSensor
    async void OnSimpleOrientationChanged(SimpleOrientationSensor sender,
                                          SimpleOrientationSensorOrientation
                                          ↪ChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                SetSimpleOrientationText(args.Orientation);
            });
    }

    void SetSimpleOrientationText(SimpleOrientation simpleOrientation)
    {
        this.simpleOrientation.Text = simpleOrientation.ToString();
    }
}
```

Poniżej pokazany jest widok ekranu tabletu używanego przeze mnie przy pisaniu tej książki, gdy został on umieszczony w stacji dokującej.



Nie przejmuj się, jeżeli zobaczysz wartość modułu odbiegającą od 1. Oznacza to tylko, że przyspieszeniomierz nie jest tak precyzyjny, jak byśmy chcieli.

Zarówno składowa *X*, jak i *Y* jest ujemna, co wskazuje, że tablet został odchylony w tył. Jak wcześniej wspomniałem, jeżeli tablet jest ustawiony pionowo w górę, teoretycznie wektor ma wartość (**0**, **–1**, **0**), a jeżeli leży na biurku ekranem w górę, teoretycznie wektor ma wartość (**0**, **0**, **–1**). Pomiędzy tymi dwoma pozycjami tablet jest obracany wokół osi *X*. Przekazując wartości *Y* i *Z* do metody `Math.Atan2`, otrzymamy kąt obrotu.

Jeżeli uruchomisz ten program na urządzeniu przenośnym, możesz poruszać nim w różnych orientacjach, aby zobaczyć efekty. Generalnie powinieneś zauważyć analogię pomiędzy wartościami z `SimpleOrientationSensor` a z `Accelerometer`:

SimpleOrientationSensor	Wektor Accelerometer
`NotRotated`	~ **(0, –1, 0)**
`Rotated90DegreesCounterClockwise`	~ **(–1, 0, 0)**
`Rotated180DegreesCounterClockwise`	~ **(0, 1, 0)**
`Rotated270DegreesCounterClockwise`	~ **(1, 0, 0)**
`Faceup`	~ **(0, 0, –1)**
`Facedown`	~ **(0, 0, 1)**

Użyty tu symbol przybliżenia (~) pokazuje, że powinniśmy traktować te wartości bardzo liberalnie. Wektor `Accelerometer` wykazuje pewne fluktuacje, zanim osiągnie wartości powodujące zmianę w `SimpleOrientationSensor`.

Ten prosty program *AccelerometerAndSimpleOrientation* nie zakłada żadnej preferowanej orientacji, więc gdy będziemy poruszać tabletem, Windows będzie automatycznie zmieniał orientację, zakładając, że nie chcesz czytać liczb do góry nogami. Powinieneś zauważyć korelację pomiędzy wartościami `SimpleOrientationSensor` a orientacją ekranu, ponieważ Windows zmienia sposób wyświetlania ekranu, bazując na tych wartościach! Jeżeli zablokujesz możliwość zmiany orientacji ekranu przez Windows (w dowolny sposób), dane wyświetlane przez program się nie zmienią.

W rzeczywistości możesz uznać, że ciągła zmiana orientacji jest irytująca. W czasie takiej zmiany aktualizacja ekranu jest na chwilę przerywana, a zawartość ekranu dopasowuje się do zmiany. Jeżeli chwilę o tym pomyślisz, prawdopodobnie dojdziesz do wniosku, że program korzystający z obiektu `Accelerometer` do zmiany zawartości ekranu powinien mieć zablokowaną automatyczną zmianę orientacji.

Z tego powodu wszystkie programy w pozostałej części rozdziału zawierają w konstruktorze programu wiersz powodujący ustawienie preferowanej orientacji:

```
DisplayProperties.AutoRotationPreferences = DisplayProperties.NativeOrientation;
```

Po uruchomieniu programu *AccelerometerAndSimpleOrientation* na urządzeniu przenośnym możesz zauważyć, że w przypadku wykonania szybkiego ruchu zmienia się kierunek i moduł wektora przyspieszenia, który przestaje wskazywać 1g. Jeżeli na przykład ruszysz urządzeniem w lewo, wektor przyspieszenia będzie wskazywał w prawo, ale wyłącznie w czasie, gdy urządzenie przyspiesza. Jeżeli uda Ci się przesuwać je ze stałą szybkością, wektor przyspieszenia wróci do stałej wartości modułu i będzie wskazywał w kierunku środka Ziemi. Nagłe zatrzymanie urządzenia spowoduje, że wektor przyspieszenia wskaże również zmianę w szybkości.

Klasa `Accelerometer` definiuje również zdarzenie o nazwie `Shaken`, które nie niesie ze sobą żadnych informacji. Zdarzenie to jest przydatne, gdy program musi „rzucić" parą kostek, zaproponować inną restaurację, usunąć rysunek lub wycofać przypadkowe usunięcie.

Jednym z częstych zastosowań klasy `Accelerometer` jest oznaczanie poziomu. W pliku XAML tworzone są cztery elementy `Ellipse`. Trzy z nich są narysowane koncentrycznie, a czwarta reprezentuje bąbelek poziomicy.

Listing 18.5. Projekt: BubbleLevel | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid Name="centeredGrid"
            HorizontalAlignment="Center"
            VerticalAlignment="Center">
        <Ellipse Name="outerCircle"
                    Stroke="{StaticResource ApplicationForegroundThemeBrush}" />

        <Ellipse Name="halfCircle"
                    Stroke="{StaticResource ApplicationForegroundThemeBrush}" />

        <Ellipse Width="24"
                    Height="24"
                    Stroke="{StaticResource ApplicationForegroundThemeBrush}" />

        <Ellipse Fill="Red"
                    Width="24"
                    Height="24"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <Ellipse.RenderTransform>
                <TranslateTransform x:Name="bubbleTranslate" />
            </Ellipse.RenderTransform>
        </Ellipse>
    </Grid>
</Grid>
```

W pliku kodu ukrytego właściwości `DisplayProperties.AutoRotationPreferences` przypisywana jest wartość `DisplayProperties.NativeOrientation`. Nie ma powodu, aby w tym programie Windows automatycznie zmieniał orientację ekranu. Program posiada również metodę obsługi zdarzenia `SizeChanged`, w którym ustawiana jest wielkość elementów `outerCircle` oraz `halfCircle`.

Listing 18.6. Projekt: BubbleLevel | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Accelerometer accelerometer = Accelerometer.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
        ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
        SizeChanged += OnMainPageSizeChanged;
    }
```

```
    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (accelerometer != null)
        {
            accelerometer.ReportInterval = accelerometer.MinimumReportInterval;
            SetBubble(accelerometer.GetCurrentReading());
            accelerometer.ReadingChanged += OnAccelerometerReadingChanged;
        }
        else
        {
            await new MessageDialog("Przyspieszeniomierz nie jest
            ↪dostępny").ShowAsync();
        }
    }

    void OnMainPageSizeChanged(object sender, SizeChangedEventArgs args)
    {
        double size = Math.Min(args.NewSize.Width, args.NewSize.Height);
        outerCircle.Width = size;
        outerCircle.Height = size;
        halfCircle.Width = size / 2;
        halfCircle.Height = size / 2;
    }

    async void OnAccelerometerReadingChanged(Accelerometer sender,
                                             AccelerometerReadingChangedEventArgs
                                             ↪args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            SetBubble(args.Reading);
        });
    }

    void SetBubble(AccelerometerReading accelerometerReading)
    {
        if (accelerometerReading == null)
            return;

        double x = accelerometerReading.AccelerationX;
        double y = accelerometerReading.AccelerationY;

        bubbleTranslate.X = -x * centeredGrid.ActualWidth / 2;
        bubbleTranslate.Y = y * centeredGrid.ActualHeight / 2;
    }
}
```

Metoda `SetBubble` jest bardzo prosta. Jej zadaniem jest pobranie składników *X* i *Y* z wektora przyspieszenia i ustawienie na ich podstawie współrzędnych `X` i `Y` środka bąbla przez skalowanie ich do promienia zewnętrznego okręgu. Zwróć uwagę na przypadek, gdy tablet leży na biurku ekranem w dół lub w górę. Składnik *Z* przyspieszenia wynosi 1 lub –1, a składniki *X* i *Y* są równe 0, co oznacza, że bąbel znajduje się na środku ekranu. Jest to prawidłowe.

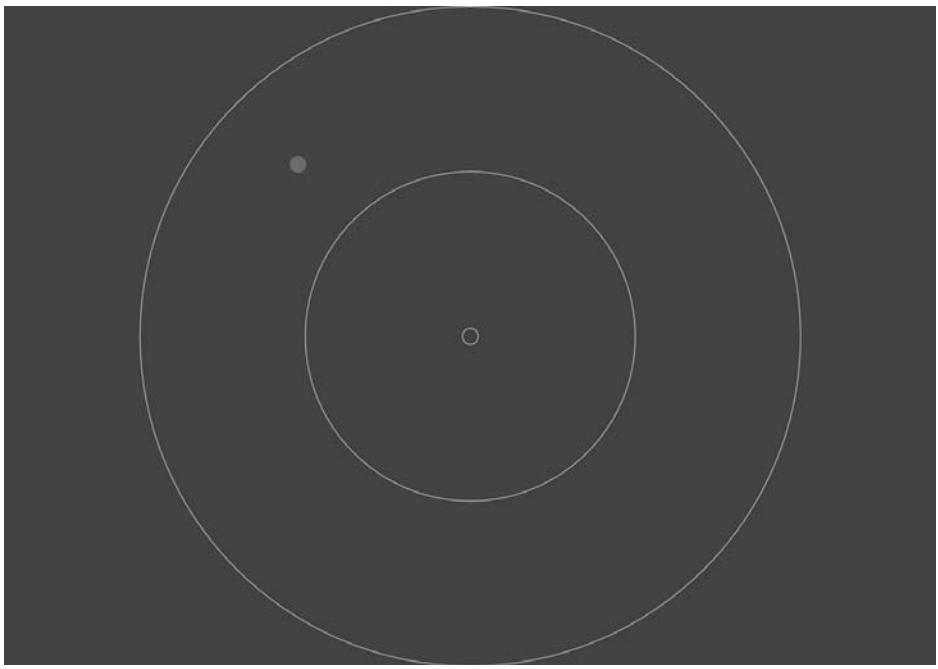
Teraz przestaw tablet tak, aby ekran był prostopadły do powierzchni Ziemi. Składowa *Z* staje się zerowa. Oznacza to, że moduł przyspieszenia zależy wyłącznie od składników *X* i *Y*. Inaczej mówiąc:

$$x^2 + y^2 = 1^2$$

Jest to równanie okręgu w dwóch wymiarach, więc bąbel znajduje się gdzieś na zewnętrznym okręgu. Miejsce, gdzie dokładnie się znajduje, zależy od obrotu tabletu względem osi *Z*.

Wektor przyspieszenia wskazuje w dół, w kierunku środka Ziemi, a bąbel przesuwa się w górę, więc musimy zamienić znaki składników *X* i *Y*, aby skonwertować je na współrzędne dwuwymiarowego ekranu. Pamiętaj jednak, że oś *Y* wektora przyspieszenia jest i tak odwrócona w stosunku do współrzędnych ekranowych, więc konieczna jest zmiana znaku wyłącznie składnika *X*, co można zauważyć w ostatnich dwóch wierszach kodu.

Program ten uruchomiony na tablecie Microsoft Surface wygląda następująco:



Oczywiście na rysunku nie widać, jak bardzo drży rysowany bąbel. Wartości z klasy `Accelerometer` są surowe i w rzeczywistej aplikacji na pewno będziesz chciał je nieco wygładzić. Zrobimy to w następnych dwóch programach.

Podążaj za kulką

Klasa `Accelerator` jest często stosowana w grach na urządzenia przenośne. Jeżeli na przykład masz grę symulującą jazdę samochodem, to użytkownik może nim kierować przez przechylanie komputera na lewo i prawo.

Poniższe dwa programy symulują kulkę toczącą się po ekranie. Jeżeli będziesz trzymał tablet równolegle do Ziemi i będziesz balansował rzeczywistą kulką położoną na powierzchni ekranu, możesz ją przetaczać przez pochylanie ekranu w dwóch płaszczyznach.

Im większe pochylenie, tym większe przyspieszenie kulki. Wirtualna kulka w dwóch następnych programach przesuwa się w podobny sposób. Podobnie jak w programie symulującym poziomicę, programy te ignorują składową *Z* z wektora `Accelerometer` i do zarządzania przyspieszeniem na dwuwymiarowej powierzchni ekranu korzystają wyłącznie ze składowych *X* oraz *Y*.

W programie *TiltAndRoll*, gdy kulka uderza o krawędź, traci całe przyspieszenie w kierunku prostopadłym do krawędzi i kontynuuje toczenie się wzdłuż niej, o ile posiada jeszcze przyspieszenie skierowane w tym kierunku. Kulka jest zdefiniowana w pliku XAML. Element `EllipseGeometry` pozwala na pozycjonowanie kulki w dowolnym miejscu poprzez ustawienie właściwości `Center`.

Listing 18.7. Projekt: TiltAndRoll | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Path Fill="Red">
        <Path.Data>
            <EllipseGeometry x:Name="ball" />
        </Path.Data>
    </Path>
</Grid>
```

Plik kodu ukrytego zaczyna się od zdefiniowania stałej `GRAVITY` określonej w pikselach na sekundę do kwadratu. Teoretycznie tocząca się bez tarcia kulka podlega sile grawitacji, ale przyspieszenie toczącej się kulki wynosi 2/3 przyspieszenia grawitacyjnego (patrz A.P. French, *Newtonian Mechanics*, W.W. Norton 1971, s. 652 – 653). Oznacza to, że można obliczyć wartość `GRAVITY` przez pomnożenie 10 metrów na sekundę do kwadratu przez 0,025 m na cal, przez 95 pikseli na cal i przez 2/3, co daje wartość około 25 000, ale na potrzeby programu znacznie ją zmniejszyłem.

Wartość wektora dwuwymiarowego jest bardzo przydatna w obliczeniach przyspieszenia w dwóch wymiarach, prędkości i położenia, więc dołączyłem do programu strukturę `Vector2` z rozdziału 13., „Dotyk itd.”.

Ponieważ kulka musi się toczyć niezależnie od wywołań zdarzenia `ReadingChanged` z obiektu `Accelerometer`, program nie instaluje metody obsługi, ale zamiast tego korzysta z `CompositionTarget.Rendering` do odczytania bieżącej wartości i zastosowania jej do ruchu kulki. Zwróć uwagę, że składniki *X* i *Y* odczytu dokonywanego w obiekcie `Accelerometer` są używane do utworzenia wartości `Vector2`, które są następnie uśredniane z poprzednimi wartościami, które z kolei były uśrednione z poprzednimi itd. Jest to niezwykle prosta metoda wygładzania.

Listing 18.8. Projekt: TiltAndRoll | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    const double GRAVITY = 5000;     // Piksele na sekundę do kwadratu
    const double BALL_RADIUS = 32;

    Accelerometer accelerometer = Accelerometer.GetDefault();
    TimeSpan timeSpan;
    Vector2 acceleration;
    Vector2 ballPosition;
    Vector2 ballVelocity;
```

```
public MainPage()
{
    this.InitializeComponent();
    DisplayProperties.AutoRotationPreferences =
    ↪DisplayProperties.NativeOrientation;

    ball.RadiusX = BALL_RADIUS;
    ball.RadiusY = BALL_RADIUS;

    Loaded += OnMainPageLoaded;
}

async void OnMainPageLoaded(object sender, RoutedEventArgs args)
{
    if (accelerometer == null)
    {
        await new MessageDialog("Przyspieszeniomierz nie jest
        ↪dostępny").ShowAsync();
    }
    else
    {
        CompositionTarget.Rendering += OnCompositionTargetRendering;
    }
}

void OnCompositionTargetRendering(object sender, object args)
{
    AccelerometerReading reading = accelerometer.GetCurrentReading();

    if (reading == null)
        return;

    // Odczyt czasu od ostatniego zdarzenia
    TimeSpan timeSpan = (args as RenderingEventArgs).RenderingTime;
    double elapsedSeconds = (timeSpan - this.timeSpan).TotalSeconds;
    this.timeSpan = timeSpan;

    // Konwersja danych przyspieszeniomierza na współrzędne wyświetlacza
    double x = reading.AccelerationX;
    double y = -reading.AccelerationY;

    // Pobranie bieżącego przyspieszenia X-Y i wygładzenie go
    acceleration = 0.5 * (acceleration + new Vector2(x, y));

    // Obliczenie nowej szybkości i pozycji
    ballVelocity += GRAVITY * acceleration * elapsedSeconds;
    ballPosition += ballVelocity * elapsedSeconds;

    // Sprawdzenie kolizji z brzegiem
    if (ballPosition.X - BALL_RADIUS < 0)
    {
        ballPosition = new Vector2(BALL_RADIUS, ballPosition.Y);
        ballVelocity = new Vector2(0, ballVelocity.Y);
    }
    if (ballPosition.X + BALL_RADIUS > this.ActualWidth)
    {
        ballPosition = new Vector2(this.ActualWidth - BALL_RADIUS, ballPosition.Y);
```

```
            ballVelocity = new Vector2(0, ballVelocity.Y);
        }
        if (ballPosition.Y - BALL_RADIUS < 0)
        {
            ballPosition = new Vector2(ballPosition.X, BALL_RADIUS);
            ballVelocity = new Vector2(ballVelocity.X, 0);
        }
        if (ballPosition.Y + BALL_RADIUS > this.ActualHeight)
        {
            ballPosition = new Vector2(ballPosition.X, this.ActualHeight -
            ↪BALL_RADIUS);
            ballVelocity = new Vector2(ballVelocity.X, 0);
        }
        ball.Center = new Point(ballPosition.X, ballPosition.Y);
    }
}
```

Dwa najważniejsze obliczenia to:

```
ballVelocity += GRAVITY * acceleration * elapsedSeconds;
ballPosition += ballVelocity * elapsedSeconds;
```

Pamiętaj, że `acceleration`, `ballVelocity` oraz `ballPosition` są wartościami `Vector2`, więc mają one składowe *X* i *Y*. Prędkość jest zwiększana o wartość `acceleration` pomnożoną o czas, a położenie jest zwiększane o prędkość pomnożoną przez czas. Pozostało tylko sprawdzać, czy nowa pozycja znajduje się poza granicami strony. Jeżeli tak, jest przesuwana na stronę i jeden z komponentów prędkości jest ustawiany na zero.

Fizyka ruchu kulki jest dosyć realistyczna. Wraz ze zwiększaniem i zmniejszaniem pochylenia zwiększa się i zmniejsza wartość przyspieszenia. Co więcej, ponieważ program korzysta ze wzorów na prędkość i położenie, dosyć łatwe jest dodanie funkcji odbijania. Prosty sposób polega na tym, że gdy kulka trafia w krawędź, można nadać przeciwny znak składnikowi prędkości, zamiast go zerować. Jednak oznacza to, że kulka po odbiciu ma taką samą wartość prędkości, co jest mało realistyczne. Sensowniejsze jest dodanie współczynnika tłumienia, który nazwałem `BOUNCE`. Program *TiltAndBounce* jest niemal identyczny z programem *TiltAndRoll*, poza stałą `BOUNCE` i inną logiką ruchu kulki w metodzie obsługi `CompositionTarget.Rendering`.

Listing 18.9. Projekt: TiltAndBounce | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    const double BOUNCE = -2.0 / 3; // Ułamek szybkości
...
    void OnCompositionTargetRendering(object sender, object args)
    {
        AccelerometerReading reading = accelerometer.GetCurrentReading();

        if (reading == null)
            return;

        // Pobranie czasu
        TimeSpan timeSpan = (args as RenderingEventArgs).RenderingTime;
        double elapsedSeconds = (timeSpan - this.timeSpan).TotalSeconds;
        this.timeSpan = timeSpan;
```

```
            // Konwersja danych przyspieszeniomierza na współrzędne wyświetlacza
            double x = reading.AccelerationX;
            double y = -reading.AccelerationY;

            // Pobranie bieżącego przyspieszenia X-Y i wygładzenie go
            acceleration = 0.5 * (acceleration + new Vector2(x, y));

            // Obliczenie nowej szybkości i pozycji
            ballVelocity += GRAVITY * acceleration * elapsedSeconds;
            ballPosition += ballVelocity * elapsedSeconds;

            // Sprawdzenie kolizji z brzegiem
            bool needAnotherLoop = true;

            while (needAnotherLoop)
            {
                needAnotherLoop = false;

                if (ballPosition.X - BALL_RADIUS < 0)
                {
                    ballPosition = new Vector2(-ballPosition.X + 2 * BALL_RADIUS,
                    ↪ballPosition.Y);
                    ballVelocity = new Vector2(BOUNCE * ballVelocity.X, ballVelocity.Y);
                    needAnotherLoop = true;
                }
                else if (ballPosition.X + BALL_RADIUS > this.ActualWidth)
                {
                    ballPosition = new Vector2(-ballPosition.X + 2 * (this.ActualWidth -
                    ↪BALL_RADIUS),
                                               ballPosition.Y);
                    ballVelocity = new Vector2(BOUNCE * ballVelocity.X, ballVelocity.Y);
                    needAnotherLoop = true;
                }
                else if (ballPosition.Y - BALL_RADIUS < 0)
                {
                    ballPosition = new Vector2(ballPosition.X, -ballPosition.Y + 2 *
                    ↪BALL_RADIUS);
                    ballVelocity = new Vector2(ballVelocity.X, BOUNCE * ballVelocity.Y);
                    needAnotherLoop = true;
                }
                else if (ballPosition.Y + BALL_RADIUS > this.ActualHeight)
                {
                    ballPosition = new Vector2(ballPosition.X,
                                               -ballPosition.Y + 2 * (this.ActualHeight -
                                               ↪BALL_RADIUS));
                    ballVelocity = new Vector2(ballVelocity.X, BOUNCE * ballVelocity.Y);
                    needAnotherLoop = true;
                }
            }
            ball.Center = new Point(ballPosition.X, ballPosition.Y);
        }
    }
```

W programie *TiltAndRoll* możliwe jest, że kulka wykroczy poza dwie sąsiednie krawędzie w tym samym zdarzeniu, ale przypadki te są obsłużone za pomocą serii instrukcji `if`. W tym programie odbijanie kulki od jednego brzegu może spowodować wyjście poza drugi brzeg, przez co niezbędna jest pętla do testowania pozycji, obsługująca wszystkie odbicia.

Dwie północe

Choć klasa `Accelerator` pozwala sprawdzić kierunek w dół, nie ujawnia ona pełnej orientacji urządzenia w przestrzeni 3D. Aby zrozumieć, co mam na myśli, uruchom program *AccelerometerAndSimpleOrientation* na urządzeniu przenośnym. Wstań i ustaw urządzenie w jakiś dziwny sposób. Dzięki klasie `Accelerometer` wiemy, gdzie jest dół. Teraz obróć całe ciało wokół osi. Tablet obróci się o 360 stopni w przestrzeni, ale `Accelerometer` będzie raportował w zasadzie taką samą wartość, ponieważ kierunek w dół pozostał niezmienny.

Co się zmienia, gdy obracamy tabletem wokół wektora przyspieszenia? Odpowiedź jest prosta: zmienia się położenie względem północy. Dlatego właśnie ważny jest sensor kompasu, `Compass`, który udostępnia nam brakujący element pozwalający na określenie orientacji tabletu. Po połączeniu odczytów z obiektów `Compass` i `Accelerometer` możemy określić kompletne położenie tabletu w przestrzeni trójwymiarowej. Możemy też pozwolić, aby zrobił to za nas system Windows.

Klasa `Compass` jest zbudowana podobnie do `Accelerometer`, a klasa `CompassReading` zawiera dwie właściwości: `HeadingMagneticNorth` oraz `HeadingTrueNorth`. Obie właściwości zawierają wartości w stopniach i wskazują kąt odchylenia położenia komputera względem północy. Jeżeli położysz tablet na powierzchni równoległej do ziemi i górę ekranu skierujesz w stronę północy (przez „górę" rozumiem kierunek dodatniej osi *Y* pokazanej na rysunku we wcześniejszej części rozdziału), kąty te powinny być bliskie zeru. Gdy będziesz obracał tabletem w kierunku wschodnim, kąty będą się zwiększać.

Oczywiście, kąty te nie powinny być takie same, poza niektórymi lokalizacjami na świecie. Tablet zawiera magnetometr, który wykrywa północ magnetyczną (która jest zależna od pola magnetycznego Ziemi), ale biegun magnetyczny nie pokrywa się z biegunem geograficznym. Co interesujące, właściwość `HeadingMagneticNorth` ma typ `double`, natomiast `HeadingTrueNorth` jest typu `double` dopuszczającego wartości `null`, co sugeruje, że wartość ta może nie być dostępna.

Wypróbujmy to. W pliku XAML projektu *SimpleCompass* zdefiniowane są dwie strzałki graficzne znajdujące się w środku ekranu i wskazujące w górę.

Listing 18.10. Projekt: SimpleCompass | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Canvas HorizontalAlignment="Center"
            VerticalAlignment="Center">
        <Path Fill="Magenta"
                Data="M -10 0 L 10 0, 10 -300, 0 -350, -10 -300 Z">
            <Path.RenderTransform>
                <RotateTransform x:Name="magNorthRotate" />
            </Path.RenderTransform>
        </Path>

        <Path Name="trueNorthPath"
                Fill="Blue"
                Data="M -10 0 L 10 0, 10 -300, 0 -350, -10 -300 Z">
            <Path.RenderTransform>
                <RotateTransform x:Name="trueNorthRotate" />
            </Path.RenderTransform>
        </Path>
    </Canvas>
</Grid>
```

Kolor magenta będzie odpowiadał północy magnetycznej, natomiast niebieski północy geograficznej.

Jeżeli wartość `HeadingTrueNorth` wynosi `null`, drugi z obiektów `Path` jest ukrywany.

Listing 18.11. Projekt: SimpleCompass | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Compass compass = Compass.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
        ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
    }

    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (compass != null)
        {
            ShowCompassValues(compass.GetCurrentReading());
            compass.ReportInterval = compass.MinimumReportInterval;
            compass.ReadingChanged += OnCompassReadingChanged;
        }
        else
        {
            await new MessageDialog("Kompas jest niedostępny").ShowAsync();
        }
    }

    async void OnCompassReadingChanged(Compass sender, CompassReadingChangedEventArgs
    ↪args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                ShowCompassValues(args.Reading);
            });
    }

    void ShowCompassValues(CompassReading compassReading)
    {
        if (compassReading == null)
            return;

        magNorthRotate.Angle = -compassReading.HeadingMagneticNorth;

        if (compassReading.HeadingTrueNorth.HasValue)
        {
            trueNorthPath.Visibility = Visibility.Visible;
            trueNorthRotate.Angle = -compassReading.HeadingTrueNorth.Value;
        }
        else
        {
            trueNorthPath.Visibility = Visibility.Collapsed;
```

```
        }
    }
}
```

Zwróć uwagę, że kąty obrotu obu strzałek są ustawiane względem ujemnych wartości właściwości `HeadingMagneticNorth` i `HeadingTrueNorth`. Wartości te wskazują na obrót komputera względem północy, więc strzałki powinny obracać się odwrotnie i wskazywać na kierunek północny względem komputera.

Na obu komputerach, z których korzystałem podczas pisania tej książki — w tym Microsoft Surface — wyniki były rozczarowujące. Na obu właściwość `HeadingTrueNorth` zawsze miała wartość `null`. Na Microsoft Surface wartość północy magnetycznej była błędna. Na tablecie Samsung wartości zmieniały się od 0 do 180 stopni! Na tablecie mojego redaktora technicznego właściwość `HeadingMagneticNorth` zawsze wynosiła 0.

Teoretycznie północ geograficzna może być wyliczona na podstawie północy magnetycznej i położenia komputera, ale włączenie w pliku *Package.appxmanifest* funkcji *Location* nie pomogło.

Możemy tylko mieć nadzieję, że w kolejnych komputerach kompas będzie działał na tyle dobrze, że pozwoli w połączeniu z danymi przyspieszeniomierza określić pełne dane o orientacji.

Inklinometr = przyspieszeniomierz + kompas

Sensor `Inclinometer` jest jedną z dwóch klas, która wewnętrznie łączy dane z przyspieszeniomierza i kompasu, wewnętrznie je wygładzając. Klasa ta udostępnia dane w postaci kątów *odchylenia*, *pochylenia* i *obrotu*, które są terminami używanymi w mechanice lotu.

Kąty odchylenia, pochylenia i obrotu są często nazywane kątami Eulera, na cześć osiemnastowiecznego matematyka Leonarda Eulera, który przedstawił matematyczny model obrotów w przestrzeni trójwymiarowej. Jeżeli lecisz samolotem, odchylenie określa kierunek, w którym jest skierowany nos samolotu. Gdy samolot skręca w lewo lub w prawo, zmienia się wartość odchylenia. Pochylenie określa kąt nosa samolotu w stosunku do poziomu. Gdy samolot się wznosi lub opada, zmienia się pochylenie. Obrót jest osiągany przez przechylanie w lewo lub prawo.

Aby zrozumieć, jak te wartości odnoszą się do komputera, możemy zwizualizować „latanie" na tablecie, jak na magicznym dywanie. Załóżmy, że siedzisz na ekranie twarzą w kierunku góry (oczywiście w natywnej orientacji tabletu) i startujesz. We wcześniej pokazanym układzie współrzędnych odchylenie jest obrotem wokół osi *Z*, pochylenie jest obrotem wokół osi *X*, a obrót zachodzi wokół osi *Y*.

Program *YawPitchRoll* pozwala przedstawić te kąty. Plik XAML zawiera kilka elementów `Rectangle` używanych dla linii, niektóre elementy `Ellipse` są wyświetlane jako kulki i nieco tekstu.

Listing 18.12. Projekt: YawPitchRoll | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <!-- Pochylenie -->
    <Rectangle Fill="Blue"
               Width="3"
               HorizontalAlignment="Center"
```

```
            VerticalAlignment="Stretch" />

<Path Name="pitchPath"
        Stroke="Blue">
    <Path.Data>
        <EllipseGeometry x:Name="pitchEllipse" RadiusX="20" RadiusY="20" />
    </Path.Data>
</Path>

<!-- Obrót -->
<Rectangle Fill="Red"
            Height="3"
            HorizontalAlignment="Stretch"
            VerticalAlignment="Center" />

<Path Name="rollPath"
        Stroke="Red"
        Fill="Red">
    <Path.Data>
        <EllipseGeometry x:Name="rollEllipse" RadiusX="20" RadiusY="20" />
    </Path.Data>
</Path>

<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <!-- Pochylenie -->
    <TextBlock Text="POCHYLENIE"
                Grid.Row="0"
                Grid.Column="0"
                Foreground="Blue"
                HorizontalAlignment="Right"
                Margin="0 0 24 0" />

    <TextBlock Name="pitchValue"
                Grid.Row="0"
                Grid.Column="1"
                Foreground="Blue"
                HorizontalAlignment="Left"
                Margin="24 0 0 0" />

    <!-- Obrót -->
    <TextBlock Text="OBRÓT"
                Grid.Row="1"
                Grid.Column="0"
                Foreground="Red"
                HorizontalAlignment="Left"
                VerticalAlignment="Top"
                Margin="0 108 0 0">
        <TextBlock.RenderTransform>
```

```
                <RotateTransform Angle="-90" />
            </TextBlock.RenderTransform>
        </TextBlock>

        <TextBlock Name="rollValue"
                   Grid.Row="0"
                   Grid.Column="0"
                   Foreground="Red"
                   HorizontalAlignment="Left"
                   VerticalAlignment="Bottom">
            <TextBlock.RenderTransform>
                <RotateTransform Angle="-90" />
            </TextBlock.RenderTransform>
        </TextBlock>

        <!-- Odchylenie -->
        <Grid Grid.Row="0"
              Grid.Column="1"
              HorizontalAlignment="Stretch"
              VerticalAlignment="Bottom"
              RenderTransformOrigin="0 1">
            <StackPanel Orientation="Horizontal"
                        HorizontalAlignment="Center">
                <TextBlock Text="ODCHYLENIE = " Foreground="Green" />
                <TextBlock Name="yawValue" Foreground="Green" />
            </StackPanel>

            <Rectangle Fill="Green"
                       Height="3"
                       HorizontalAlignment="Stretch"
                       VerticalAlignment="Bottom" />

            <Grid.RenderTransform>
                <TransformGroup>
                    <RotateTransform Angle="-90" />
                    <RotateTransform x:Name="yawRotate" />
                </TransformGroup>
            </Grid.RenderTransform>
        </Grid>
    </Grid>
</Grid>
```

Jak można zauważyć w pliku kodu ukrytego, klasa `Inclinometer` jest tworzona podobnie jak `Accelerometer` i `Compass`.

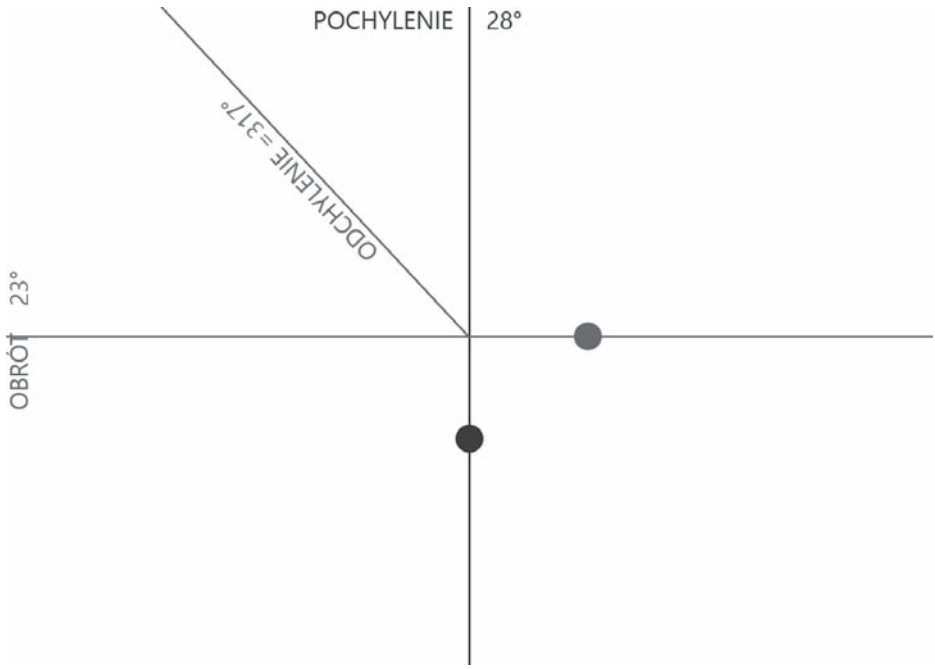
Listing 18.13. Projekt: YawPitchRoll | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Inclinometer inclinometer = Inclinometer.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
        ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
```

```
}

async void OnMainPageLoaded(object sender, RoutedEventArgs args)
{
    if (inclinometer == null)
    {
        await new MessageDialog("Nie można znaleźć inklinometru").ShowAsync();
    }
    else
    {
        ShowYawPitchRoll(inclinometer.GetCurrentReading());
        inclinometer.ReportInterval = inclinometer.MinimumReportInterval;
        inclinometer.ReadingChanged += OnInclinometerReadingChanged;
    }
}

async void OnInclinometerReadingChanged(Inclinometer sender,
                                        InclinometerReadingChangedEventArgs args)
{
    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            ShowYawPitchRoll(args.Reading);
        });
}

void ShowYawPitchRoll(InclinometerReading inclinometerReading)
{
    if (inclinometerReading == null)
        return;

    double yaw = inclinometerReading.YawDegrees;
    double pitch = inclinometerReading.PitchDegrees;
    double roll = inclinometerReading.RollDegrees;

    yawValue.Text = yaw.ToString("F0") + "°";
    pitchValue.Text = pitch.ToString("F0") + "°";
    rollValue.Text = roll.ToString("F0") + "°";

    yawRotate.Angle = yaw;

    if (pitch <= 90 && pitch >= -90)
    {
        pitchPath.Fill = pitchPath.Stroke;
        pitchEllipse.Center = new Point(this.ActualWidth / 2,
                                        this.ActualHeight * (pitch + 90) / 180);
    }
    else
    {
        pitchPath.Fill = null;

        if (pitch > 90)
            pitchEllipse.Center = new Point(this.ActualWidth / 2,
                                            this.ActualHeight * (270 - pitch) /
                                            ↪180);
        else // Pochylenie < -90
            pitchEllipse.Center = new Point(this.ActualWidth / 2,
                                            this.ActualHeight * (-90 - pitch) /
                                            ↪180);
```

```
        }
        rollEllipse.Center = new Point(this.ActualWidth * (roll + 90) / 180,
                                       this.ActualHeight / 2);
    }
}
```

Nie istnieje sekretne źródło danych kompasu dla inklinometru. Właściwość `YawDegrees` ma takie same ograniczenia (lub błędy) jak `Compass`, poza tym że się wzajemnie kompensują — suma wartości `YawDegrees` i odczytu z `Compass` jest zawsze równa 360. Gdy tablet leży na płaskiej powierzchni ekranem do góry, linia odchylenia wskazuje na północ (mniej więcej), a kulki pochylenia i odchylenia znajdują się na środku ekranu. Gdy będziesz pochylał tablet w górę lub w dół, `PitchDegrees` zmienia się od 90 stopni, gdy tablet stoi pionowo w górę, do –90, gdy tablet stoi pionowo w dół. `RollDegrees` zmienia się od 90 do –90 stopni, gdy tablet pochyla się w lewo lub prawo. Poniższy ekran uzyskałem, podnosząc lewą i górną krawędź tabletu.



Gdy ekran jest skierowany w dół, `YawDegrees` wskazuje na południe, a `PitchDegrees` przyjmuje wartości od 90 do 180 stopni oraz od –90 do –180 stopni. Program symbolizuje te wartości za pomocą pustego, czerwonego okręgu.

Jeżeli pracujesz nad programem, gdzie obiekt lata po ekranie, te kąty Eulera mogą być wystarczające. Jednak możesz potrzebować czegoś bardziej zorientowanego matematycznie. Do tego służy następna omawiana klasa.

OrientationSensor = przyspieszeniomierz + kompas

Obroty w przestrzeni trójwymiarowej można reprezentować na kilka rodzajów, które można pomiędzy sobą konwertować. Klasa `OrientationSensor` jest bardzo podobna do `Inclinometer` w tym sensie, że łączy dane z przyspieszeniomierza i kompasu, aby zapewnić pełną orientację w przestrzeni 3D. `OrientationSensor` zapewnia tę orientację dzięki dwóm klasom:

- `SensorQuaternion`,
- `SensorRotationMatrix`.

Kwaterniony są bardzo interesujące z punktu widzenia matematyki. Podobnie jak liczby urojone pozwalają na reprezentowanie obrotów w przestrzeni dwuwymiarowej, kwaterniony reprezentują obroty w przestrzeni trójwymiarowej. Szczególnie programiści gier lubią reprezentować obroty jako kwaterniony, ponieważ mogą być one płynnie interpolowane (kwaterniony przedstawiałem w rozdziale 8. mojej książki *3D Programming for Windows*, Microsoft Press 2007).

Macierz obrotu jest zwykłą macierzą transformacji bez ostatniego wiersza i kolumny. Zwykła trójwymiarowa macierz transformacji ma 4 wiersze i 4 kolumny. W klasie `Sensor↪RotationMatrix` są zdefiniowane 3 wiersze i 3 kolumny. Taka macierz nie jest w stanie reprezentować przesunięcia czy perspektywy, a zgodnie z konwencją nie zapewnia skalowania i pochylania. Może być jednak łatwo używana do obracania obiektów w przestrzeni 3D.

Na tablecie Samsung, którego używałem w czasie pisania tej książki, macierz w `Sen↪sorRotationMatrix` zawierała same zera, więc żaden z programów wykorzystujących tę macierz nie działał. Lepsze wyniki uzyskałem w przypadku Microsoft Surface.

Gdy korzystasz z macierzy obrotu, zmiana perspektywy może być pomocna. Wyjaśniałem już, jak wartości z klas `Accelerometer` oraz `Compass` odnoszą się do układu współrzędnych 3D, który prezentowałem we wcześniejszej części rozdziału. Przy korzystaniu z macierzy obrotu z klasy `OrientationSensor` pomocne jest zwizualizowanie dwóch układów współrzędnych, jednego dla urządzenia, a drugiego dla Ziemi:

- w trójwymiarowym układzie współrzędnych dodatnia oś *Y* wskazuje na górę ekranu, dodatnia oś *X* wskazuje w prawo, a dodatnia oś *Z* wychodzi z ekranu, tak jak pokazałem wcześniej,
- w ziemskim układzie współrzędnych dodatnia oś *Y* wskazuje na północ, dodatnia oś *X* wskazuje na wschód, a dodatnia oś *Z* wychodzi z gruntu.

Te dwa układy współrzędnych są takie same, gdy komputer leży na poziomej powierzchni z ekranem wskazującym w górę i gdy górna krawędź ekranu wskazuje na północ. `SensorRotationMatrix` staje się (teoretycznie) macierzą jednostkową — jedynki znajdują się na przekątnej, a w pozostałych komórkach znajdują się zera. W przeciwnym razie macierz opisuje, w jaki sposób Ziemia jest obrócona w stosunku do komputera, co jest przeciwne do obrotu opisywanego przez kąty Eulera.

Różnica ta jest przedstawiona w programie *AxisAngleRotation*, który korzysta z kolejnej metody reprezentowania obrotu w przestrzeni trójwymiarowej — jako obrotu wokół wektora 3D. Plik XAML jest niezbyt interesującym zbiorem elementów `TextBlock`, z których część to etykiety, a część oczekuje na tekst.

Listing 18.14. Projekt: AxisAngleRotation | Plik: MainPage.xaml (fragment)

```
<Page ... >

    <Page.Resources>
        <Style x:Key="DefaultTextBlockStyle" TargetType="TextBlock">
            <Setter Property="FontFamily" Value="Lucida Sans Unicode" />
            <Setter Property="FontSize" Value="36" />
            <Setter Property="Margin" Value="0 0 48 0" />
        </Style>

        <Style x:Key="rightText" TargetType="TextBlock"
               BasedOn="{StaticResource DefaultTextBlockStyle}">
            <Setter Property="TextAlignment" Value="Right" />
            <Setter Property="Margin" Value="48 0 0 0" />
        </Style>
    </Page.Resources>

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Center"
                    VerticalAlignment="Center">

            <!-- Element Grid pokazujący odchylenie, obrót i pochylenie -->
            <Grid HorizontalAlignment="Center">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>

                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                </Grid.ColumnDefinitions>

                <Grid.Resources>
                    <Style TargetType="TextBlock"
                           BasedOn="{StaticResource DefaultTextBlockStyle}" />
                </Grid.Resources>

                <TextBlock Text="Odchylenie: " Grid.Row="0" Grid.Column="0" />
                <TextBlock Name="pitchText" Grid.Row="0" Grid.Column="1"
                           Style="{StaticResource rightText}" />

                <TextBlock Text="Obrót: " Grid.Row="1" Grid.Column="0" />
                <TextBlock Name="rollText" Grid.Row="1" Grid.Column="1"
                           Style="{StaticResource rightText}" />

                <TextBlock Text="Pochylenie: " Grid.Row="2" Grid.Column="0" />
                <TextBlock Name="yawText" Grid.Row="2" Grid.Column="1"
                           Style="{StaticResource rightText}" />
            </Grid>

            <!-- Element Grid na RotationMatrix -->
            <Grid HorizontalAlignment="Center"
                  Margin="0 48">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
```

```
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>

            <Grid.Resources>
                <Style TargetType="TextBlock"
                       BasedOn="{StaticResource rightText}" />
            </Grid.Resources>

            <TextBlock Name="m11Text" Grid.Row="0" Grid.Column="0" />
            <TextBlock Name="m12Text" Grid.Row="0" Grid.Column="1" />
            <TextBlock Name="m13Text" Grid.Row="0" Grid.Column="2" />

            <TextBlock Name="m21Text" Grid.Row="1" Grid.Column="0" />
            <TextBlock Name="m22Text" Grid.Row="1" Grid.Column="1" />
            <TextBlock Name="m23Text" Grid.Row="1" Grid.Column="2" />

            <TextBlock Name="m31Text" Grid.Row="2" Grid.Column="0" />
            <TextBlock Name="m32Text" Grid.Row="2" Grid.Column="1" />
            <TextBlock Name="m33Text" Grid.Row="2" Grid.Column="2" />
        </Grid>

        <!-- Wyświetlanie osi i kąta obrotu -->
        <Grid HorizontalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>

            <Grid.Resources>
                <Style TargetType="TextBlock" BasedOn="{StaticResource
                ➥DefaultTextBlockStyle}" />
            </Grid.Resources>

            <TextBlock Text="Kąt:" Grid.Row="0" Grid.Column="0" />
            <TextBlock Name="angleText" Grid.Row="0" Grid.Column="1"
            ➥TextAlignment="Center"/>
            <TextBlock Text="Osie:" Grid.Row="1" Grid.Column="0" />
            <TextBlock Name="axisText" Grid.Row="1" Grid.Column="1"
            ➥TextAlignment="Center" />
        </Grid>
    </StackPanel>
</Grid>
</Page>
```

W pliku kodu ukrytego tworzony jest obiekt `Inclinometer` w celu pobrania kątów odchylenia, nachylenia i obrotu oraz `OrientationSensor` w celu uzyskania (i wyświetlenia) macierzy obrotu i jej konwersji na osie (kąty) obrotu.

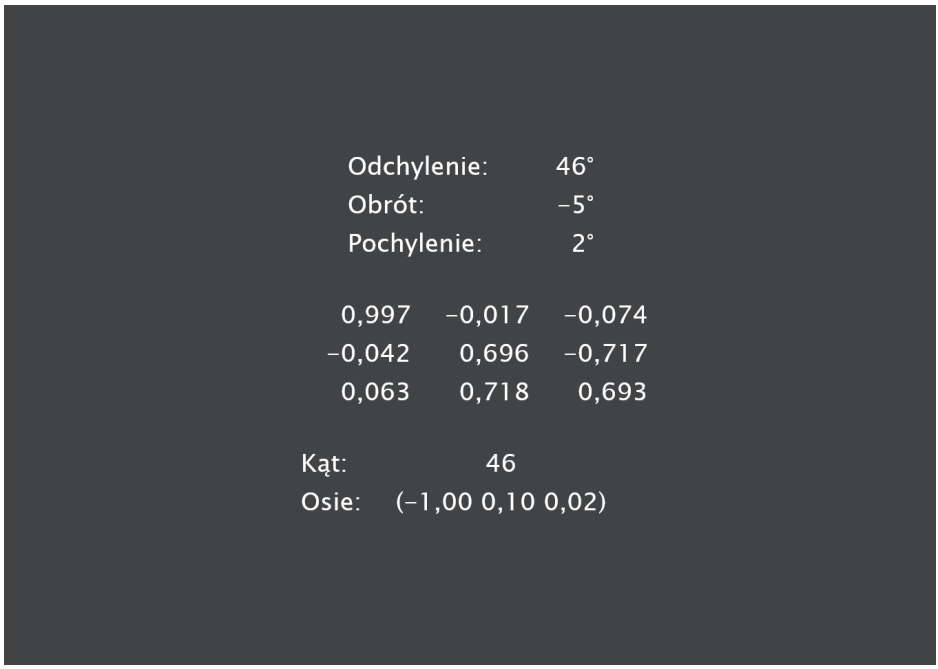
Listing 18.15. Projekt: AxisAngleRotation | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    Inclinometer inclinometer = Inclinometer.GetDefault();
    OrientationSensor orientationSensor = OrientationSensor.GetDefault();

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
        ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
    }

    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (inclinometer == null)
        {
            await new MessageDialog("Inklinometr nie jest dostępny").ShowAsync();
        }
        else
        {
            // Uruchomienie zdarzeń inklinometru
            ShowYawPitchRoll(inclinometer.GetCurrentReading());
            inclinometer.ReadingChanged += OnInclinometerReadingChanged;
        }

        if (orientationSensor == null)
        {
            await new MessageDialog("OrientationSensor nie jest dostępny").ShowAsync();
        }
        else
        {
            // Uruchomienie zdarzeń OrientationSensor
            ShowOrientation(orientationSensor.GetCurrentReading());
            orientationSensor.ReadingChanged += OrientationSensorChanged;
        }
    }

    async void OnInclinometerReadingChanged(Inclinometer sender,
                                            InclinometerReadingChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                ShowYawPitchRoll(args.Reading);
            });
    }

    void ShowYawPitchRoll(InclinometerReading inclinometerReading)
    {
        if (inclinometerReading == null)
            return;
```

```
            yawText.Text = inclinometerReading.YawDegrees.ToString("F0") + "°";
            pitchText.Text = inclinometerReading.PitchDegrees.ToString("F0") + "°";
            rollText.Text = inclinometerReading.RollDegrees.ToString("F0") + "°";
        }

        async void OrientationSensorChanged(OrientationSensor sender,
                                            OrientationSensorReadingChangedEventArgs args)
        {
            await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
                {
                    ShowOrientation(args.Reading);
                });
        }

        void ShowOrientation(OrientationSensorReading orientationReading)
        {
            if (orientationReading == null)
                return;

            SensorRotationMatrix matrix = orientationReading.RotationMatrix;

            if (matrix == null)
                return;

            m11Text.Text = matrix.M11.ToString("F3");
            m12Text.Text = matrix.M12.ToString("F3");
            m13Text.Text = matrix.M13.ToString("F3");

            m21Text.Text = matrix.M21.ToString("F3");
            m22Text.Text = matrix.M22.ToString("F3");
            m23Text.Text = matrix.M23.ToString("F3");

            m31Text.Text = matrix.M31.ToString("F3");
            m32Text.Text = matrix.M32.ToString("F3");
            m33Text.Text = matrix.M33.ToString("F3");

            // Konwersja macierzy obrotu na osie i kąt
            double angle = Math.Acos((matrix.M11 + matrix.M22 + matrix.M33 - 1) / 2);
            angleText.Text = (180 * angle / Math.PI).ToString("F0");

            if (angle != 0)
            {
                double twoSine = 2 * Math.Sin(angle);
                double x = (matrix.M23 - matrix.M32) / twoSine;
                double y = (matrix.M31 - matrix.M13) / twoSine;
                double z = (matrix.M12 - matrix.M21) / twoSine;

                axisText.Text = String.Format("({0:F2} {1:F2} {2:F2})", x, y, z);
            }
        }
    }
```

Poniżej przedstawiony jest ekran Microsoft Surface pokazujący trzy kąty Eulera na górze, macierz obrotu pośrodku oraz uzyskane osie (kąty) obrotu na dole.

Tworząc ten zrzut ekranu, trzymałem tablet skierowany w kierunku północnym, więc kąt odchylenia jest niemal równy zeru. Tablet był pochylony lekko w lewą stronę, przez



co kąt obrotu jest ujemny. Góra ekranu była podniesiona do 46 stopni. Ten sam kąt jest wyświetlany na dole, ponieważ jest pobrany z macierzy obrotu. Spójrz na osie — jest to niemal wektor (**–1**, **0**, **0**), który odpowiada ujemnej osi *X*. Korzystając z reguły prawej dłoni, wskaż kciukiem w kierunku ujemnej osi *X*. Obrót palców wskazuje, że obrót następuje w kierunku dodatnich kątów (i tak jest), więc potwierdza to, co napisałem — macierz obrotu opisuje obrót Ziemi względem komputera.

Oznacza to, że jeżeli chcesz użyć macierzy obrotu reprezentującej obrót komputera względem Ziemi, musisz odwrócić tę macierz. Klasa `SensorRotationMatrix` nie ma możliwości wykonania tej operacji, ale struktura `Matrix3D` może to zrobić (jak pamiętasz, klasa `Matrix3D` jest zdefiniowana w przestrzeni nazw `Windows.UI.Xaml.Media.Media3D` i używana w połączeniu z `Matrix3DProjection`). Wystarczy utworzyć obiekt Matrix3D na podstawie `SensorRotationMatrix` i odwrócić go.

Zamierzam użyć tej techniki do utworzenia kolejnej reprezentacji orientacji w przestrzeni trójwymiarowej.

Azymut i wysokość

Można przyjąć, że żyjemy w sferze niebieskiej. Jeżeli chcesz opisać lokalizację obiektu w przestrzeni 3D względem nas, gdy odległość nie ma znaczenia, punkty względne do tej sfery niebieskiej są bardzo wygodne. Ta sfera niebieska szczególnie dobrze nadaje się dla programów, które pozwalają używać komputera do oglądania świata, rzeczywistości wirtualnej lub rzeczywistości rozszerzonej. W takich programach trzymamy tablet tak, jakbyśmy chcieli wykonać zdjęcie za pomocą jego tylnej części, ale to, co widzimy na

ekranie, jest generowane (w całości lub w części) przez program bazujący na orientacji ekranu. Przesuwając tablet po łuku, można obserwować na ekranie odwzorowanie wycinka otaczającego nas świata.

Sfera niebieska ma analogię w ziemskiej rzeczywistości. Gdy musimy określić lokalizację na Ziemi, wykonujemy to z użyciem długości i szerokości geograficznej, które są kątami z wierzchołkiem w środku Ziemi. Dzielimy kulę ziemską na połowy wzdłuż równika. Linie szerokości geograficznej są równoległe do równika i określają dodatni kąt w stosunku do równika (z maksymalną wartością 90 stopni na biegunie północnym) w kierunku północnym oraz ujemny kąt w stosunku do równika w kierunku południowym (aż do –90 stopni na biegunie południowym). Kąty długości bazują na okręgach przechodzących przez oba bieguny i są mierzone w stosunku do południka zerowego, który przechodzi przez Greenwich w Wielkiej Brytanii.

Punkty na sferze niebieskiej możemy opisać mniej więcej w ten sam sposób, ale jesteśmy w środku tej sfery, więc terminologia jest inna.

Wskaż wyprostowanym ramieniem w dowolnym kierunku. W jaki sposób możemy zidentyfikować tę lokalizację? Na początek przesuń ramię w górę lub w dół, aby było w położeniu poziomym — czyli równoległym do powierzchni Ziemi. Kąt, o który przesunąłeś ramię w czasie tego ruchu, jest nazywany *wysokością*.

Dodatnie wartości wysokości znajdują się ponad horyzontem, a ujemne poniżej horyzontu. Pionowo w górę znajduje się *zenit*, dla którego wysokość wynosi 90 stopni. Pionowo w dół znajduje się *nadir*, dla którego wysokość wynosi –90 stopni.

Nadal wskazujesz wyprostowanym ramieniem w kierunku horyzontu, prawda? Teraz przesuń ramię tak, aby wskazywało północ. Kąt, o który przesunąłeś ramię w czasie tego ruchu, jest nazywany *azymutem*.

Połączenie wysokości i azymutu tworzy *współrzędne horyzontalne*, nazwane w ten sposób, ponieważ horyzont dzieli sferę niebieską na pół — podobnie jak równik we współrzędnych geograficznych.

Współrzędne horyzontalne nie niosą żadnych danych na temat odległości. Gdy jest zaćmienie Słońca, Słońce i Księżyc mają te same współrzędne horyzontalne. Współrzędna horyzontalna nie jest lokalizacją w przestrzeni 3D — jest kierunkiem w przestrzeni 3D od obserwatora. W tym sensie współrzędne horyzontalne są podobne do wektora 3D, ale wektor jest wyrażany we współrzędnych kwadratowych, a współrzędne horyzontalne są sferyczne.

Aby nieco ułatwić zadanie określania współrzędnych horyzontalnych, na początek zdefiniujemy strukturę `Vector3` opisującą wektor trójwymiarowy.

Listing 18.16. Projekt: EarthlyDelights | Plik: Vector3.cs

```
using System;
using Windows.Foundation;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Media.Media3D;

namespace Petzold.Windows8.VectorDrawing
{
    public struct Vector3
    {
        // Konstruktor
        public Vector3(double x, double y, double z)
            : this()
```

```
{
    X = x;
    Y = y;
    Z = z;
}

// Właściwości
public double X { private set; get; }
public double Y { private set; get; }
public double Z { private set; get; }

public double LengthSquared
{
    get { return X * X + Y * Y + Z * Z; }
}

public double Length
{
    get { return Math.Sqrt(LengthSquared); }
}

public Vector3 Normalized
{
    get
    {
        double length = this.Length;

        if (length != 0)
        {
            return new Vector3(this.X / length,
                               this.Y / length,
                               this.Z / length);
        }
        return new Vector3();
    }
}

// Właściwości statyczne
public static Vector3 UnitX
{
    get { return new Vector3(1, 0, 0); }
}

public static Vector3 UnitY
{
    get { return new Vector3(0, 1, 0); }
}

public static Vector3 UnitZ
{
    get { return new Vector3(0, 0, 1); }
}

// Metody statyczne
public static Vector3 Cross(Vector3 v1, Vector3 v2)
{
    return new Vector3(v1.Y * v2.Z - v1.Z * v2.Y,
```

```
                        v1.Z * v2.X - v1.X * v2.Z,
                        v1.X * v2.Y - v1.Y * v2.X);
}

public static double Dot(Vector3 v1, Vector3 v2)
{
    return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z;
}

public static double AngleBetween(Vector3 v1, Vector3 v2)
{
    return 180 / Math.PI * Math.Acos(Vector3.Dot(v1, v2) /
                                     v1.Length * v2.Length);
}

public static Vector3 Transform(Vector3 v, Matrix3D m)
{
    double x = m.M11 * v.X + m.M21 * v.Y + m.M31 * v.Z + m.OffsetX;
    double y = m.M12 * v.X + m.M22 * v.Y + m.M32 * v.Z + m.OffsetY;
    double z = m.M13 * v.X + m.M23 * v.Y + m.M33 * v.Z + m.OffsetZ;
    double w = m.M14 * v.X + m.M24 * v.Y + m.M34 * v.Z + m.M44;
    return new Vector3(x / w, y / w, z / w);
}

// Operatory
public static Vector3 operator +(Vector3 v1, Vector3 v2)
{
    return new Vector3(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z);
}

public static Vector3 operator -(Vector3 v1, Vector3 v2)
{
    return new Vector3(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z);
}

public static Vector3 operator *(Vector3 v, double d)
{
    return new Vector3(d * v.X, d * v.Y, d * v.Z);
}

public static Vector3 operator *(double d, Vector3 v)
{
    return new Vector3(d * v.X, d * v.Y, d * v.Z);
}

public static Vector3 operator /(Vector3 v, double d)
{
    return new Vector3(v.X / d, v.Y / d, v.Z / d);
}

public static Vector3 operator -(Vector3 v)
{
    return new Vector3(-v.X, -v.Y, -v.Z);
}

// Metody przesłaniające
public override string ToString()
{
```

```
            return String.Format("({0} {1} {2})", X, Y, Z);
        }
    }
}
```

W strukturze tej występuje dużo ułatwień, w tym tradycyjny iloraz skalarny oraz iloczyn wektorowy, jak również metoda `Transform`, która mnoży wartość `Vector3` przez wartość `Matrix3D`. W praktyce ta wartość `Matrix3D` będzie prawdopodobnie reprezentowała obrót, więc mnożenie w efekcie obróci wektor w przestrzeni trójwymiarowej.

Gdy trzymamy tablet pionowo i patrzymy na ekran, patrzymy w kierunku względnym do układu współrzędnych komputera, a dokładniej — w kierunku wektora wychodzącego z tyłu ekranu, który jest ujemną osią *Z* lub (**0**, **0**, **–1**). Musimy skonwertować tę wartość na współrzędne horyzontalne.

Utwórzmy wartość `Matrix3D` o nazwie `matrix`, bazując na obiekcie `SensorRotation` ➥`Matrix` dostarczanym przez `OrientationSensor`. Wartość ta może być odwrócona, aby reprezentowała transformację z układu współrzędnych na komputerze na ziemski układ współrzędnych:

```
matrix.Invert();
```

W celu przekształcenia wektora (**0**, **0**, **–1**) (który jest odwrotnością statycznej właściwości `UnitZ` udostępnianej przez strukturę `Vector3`) na współrzędne ziemskie użyjemy wartości `matrix`:

```
Vector3 vector = Vector3.Transform(-Vector3.UnitZ, matrix);
```

Wektor ten zawiera wartość we współrzędnych prostokątnych, więc musimy je skonwertować na współrzędne horyzontalne. Przypomnij sobie, że w ziemskim układzie współrzędnych oś *Z* wskazuje w górę. Jeżeli tablet jest trzymany pionowo, oś wychodząca z tyłu urządzenia przekształcona na współrzędne ziemskie ma składnik *Z* równy zero. Oznacza to, że azymut można wyliczyć za pomocą dobrze znanej konwersji z dwuwymiarowych współrzędnych kartezjańskich na współrzędne kątowe, a jednocześnie z radianów na stopnie:

```
double azimuth = 180 * Math.Atan2(vector.X, vector.Y) / Math.PI;
```

Wzór ten jest poprawny niezależnie od składnika *Z* przekształcanego wektora. Ponieważ wysokość przyjmuje wartości od minus do plus 90 stopni, może być wyliczona za pomocą funkcji arcus sinus:

```
double altitude = 180 * Math.Asin(vector.Z) / Math.PI;
```

Jednak o czymś zapomnieliśmy. Skonwertowaliśmy trójwymiarową macierz obrotu na współrzędne, które mają tylko dwa składniki, ponieważ ograniczyliśmy wnętrze sfery do jej powierzchni. Co się stanie, gdy wskażemy tabletem na punkt w sferze niebieskiej, a następnie obrócimy tablet wokół tej osi? Jest to ta sama wysokość i azymut, ale widok na ekranie komputera powinien się zmieniać w czasie obrotu. Brakujący element jest czasami nazywany *nachyleniem*. To nieco trudniejsze do wyliczenia, ale jest to pokazane w strukturze `HorizontalCoordinate`.

Listing 18.17. Projekt: EarthlyDelights | Plik: HorizontalCoordinate.cs

```
using System;
using Windows.UI.Xaml.Media.Media3D;
```

```
namespace Petzold.Windows8.VectorDrawing
{
    public struct HorizontalCoordinate
    {
        public HorizontalCoordinate(double azimuth, double altitude, double tilt)
            : this()
        {
            this.Azimuth = azimuth;
            this.Altitude = altitude;
            this.Tilt = tilt;
        }

        public HorizontalCoordinate(double azimuth, double altitude)
            : this(azimuth, altitude, 0)
        {
        }

        // Na wschód od północy
        public double Azimuth { private set; get; }

        public double Altitude { private set; get; }

        public double Tilt { private set; get; }

        public static HorizontalCoordinate FromVector(Vector3 vector)
        {
            double altitude = 180 * Math.Asin(vector.Z) / Math.PI;
            double azimuth = 180 * Math.Atan2(vector.X, vector.Y) / Math.PI;

            return new HorizontalCoordinate(azimuth, altitude);
        }

        public static HorizontalCoordinate FromMotionMatrix(Matrix3D matrix)
        {
            // Odwrócenie macierzy
            matrix.Invert();

            // Transformacja (0, 0, -1) - wektor wychodzący z obiektywu
            Vector3 zAxisTransformed = Vector3.Transform(-Vector3.UnitZ, matrix);

            // Odczyt współrzędnych horyzontalnych
            HorizontalCoordinate horzCoord = FromVector(zAxisTransformed);

            // Określenie teoretycznych HorizontalCoordinate dla przekształconego wektora +Y,
            //    jeżeli urządzenie jest trzymane pionowo
            double yUprightAltitude = 0;
            double yUprightAzimuth = 0;

            if (horzCoord.Altitude > 0)
            {
                yUprightAltitude = 90 - horzCoord.Altitude;
                yUprightAzimuth = 180 + horzCoord.Azimuth;
            }
            else
            {
                yUprightAltitude = 90 + horzCoord.Altitude;
                yUprightAzimuth = horzCoord.Azimuth;
```

```
            }
            Vector3 yUprightVector =
                new HorizontalCoordinate(yUprightAzimuth, yUprightAltitude).ToVector();

            // Określenie rzeczywistego przekształconego wektora +Y
            Vector3 yAxisTransformed = Vector3.Transform(Vector3.UnitY, matrix);

            // Pobranie kąta pomiędzy wektorem +Y a rzeczywistym, przekształconym wektorem +Y
            double dotProduct = Vector3.Dot(yUprightVector, yAxisTransformed);
            Vector3 crossProduct = Vector3.Cross(yUprightVector, yAxisTransformed);
            crossProduct = crossProduct.Normalized;

            // Czasami dotProduct jest nieco większy od 1, co
            // generuje wyjątek w obliczeniach angleBetween, więc...
            dotProduct = Math.Min(dotProduct, 1);
            double angleBetween = 180 * Vector3.Dot(zAxisTransformed, crossProduct)
                                      * Math.Acos(dotProduct) / Math.PI;
            horzCoord.Tilt = angleBetween;

            return horzCoord;
        }

        public Vector3 ToVector()
        {
            double x = Math.Cos(Math.PI * this.Altitude / 180) *
                       Math.Sin(Math.PI * this.Azimuth / 180);
            double y = Math.Cos(Math.PI * this.Altitude / 180) *
                       Math.Cos(Math.PI * this.Azimuth / 180);
            double z = Math.Sin(Math.PI * this.Altitude / 180);

            return new Vector3((float)x, (float)y, (float)z);
        }

        public override string ToString()
        {
            return String.Format("Azy: {0} Wys: {1} Poch: {2}",
                                 this.Azimuth, this.Altitude, this.Tilt);
        }
    }
}
```

Po wykonaniu tej konwersji jesteś na dobrej drodze do napisania programu astronomicznego, który będzie wyświetlać określony fragment nocnego nieba w zależności od orientacji ekranu, podobnie jak zrobiłem to dla Windows Phone 7.5 we wrześniowym numerze „MSDN Magazine" z 2012 roku. Zróbmy teraz coś mniej ambitnego.

Co możemy zrobić, aby obejrzeć mapę bitową, która jest znacznie większa od ekranu komputera, a nie chcemy jej zmniejszać? Tradycyjnym rozwiązaniem jest użycie pasków przewijania. Nieco nowocześniejsze rozwiązanie pozwala na przesuwanie obrazu palcami.

Jednak inne podejście polega na umieszczeniu mapy bitowej na powierzchni sfery niebieskiej. Obraz ten można oglądać przez trzymanie tabletu przed sobą i zmianę orientacji ekranu. Oczywiście nie chcemy rozciągać mapy bitowej, aby pasowała do powierzchni sfery. Zamiast tego użyjemy azymutu do przewijania w poziomie i wysokości do przewijania w pionie.

Program *EarthlyDelights* pozwala na oglądanie dużej (7793 na 4409 piksele) mapy bitowej z 500-letnim obrazem Hieronima Boscha, *Ogród ziemskich rozkoszy*. Program pobiera ten obraz z Wikipedii. Poniżej przedstawiony jest fragment wyświetlany za pomocą programu uruchomionego na Microsoft Surface:

Program nie posiada interfejsu dotykowego do skanowania lub zmiany rozmiaru obrazu. Wszystko bazuje na zmianie orientacji ekranu. Jeżeli jednak stukniesz ekran, program wykona skalowanie, aby pokazać cały obraz z zaznaczonym prostokątem pokazującym obszar wyświetlany w zwykłym trybie.

Funkcja ta nieco komplikuje program, ale uznałem ją za niezbędną.

Najważniejszą częścią pliku XAML jest oczywiście element `Image`. Zwróć uwagę, że właściwość `Stretch` elementu `Image` ma wartość `None` oraz zawiera obiekt `BitmapImage` bez ustawionego adresu URI źródła (na razie). Element `Grid` zawierający `Image` znajduje się w obiekcie `Canvas`, więc nie będzie przycięty, gdy będzie większy od ekranu (a na pewno będzie).

Listing 18.18. Projekt: EarthlyDelights | Plik: MainPage.xaml (fragment)

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <!-- Dwa elementy wyświetlane wyłącznie w czasie pobierania pliku -->
    <ProgressBar Name="progressBar"
                 VerticalAlignment="Center"
                 Margin="96 0" />

    <TextBlock Name="statusText"
               Text="ładowanie zdjęcia..."
               HorizontalAlignment="Center"
               VerticalAlignment="Center" />
    <Canvas>
        <Grid>
            <Image Stretch="None">
                <Image.Source>
                    <BitmapImage x:Name="bitmapImage"
                                 DownloadProgress="OnBitmapImageDownloadProgress"
                                 ImageFailed="OnBitmapImageFailed"
                                 ImageOpened="OnBitmapImageOpened" />
                </Image.Source>
            </Image>

            <Border Name="outlineBorder"
                    BorderBrush="White"
                    HorizontalAlignment="Left"
                    VerticalAlignment="Top">

                <Rectangle Name="outlineRectangle"
                           Stroke="Black" />

                <Border.RenderTransform>
                    <CompositeTransform x:Name="borderTransform" />
                </Border.RenderTransform>
            </Border>

            <Grid.RenderTransform>
                <CompositeTransform x:Name="imageTransform" />
            </Grid.RenderTransform>
        </Grid>
    </Canvas>

    <TextBlock Name="titleText"
               Margin="2 "/>
</Grid>
```

Element `Border` z zagnieżdżonym `Rectangle` jest używany w widoku przeskalowanym do pokazania części obrazu, która normalnie zajmuje cały ekran, ale możesz widzieć ten prostokąt również w widoku normalnym. Zewnętrzny element `CompositeTransform`

odnosi się zarówno do `Image`, jak i `Border`. W widoku normalnym transformacja ta nie wykonuje żadnych operacji. Wewnętrzny element `CompositeTransform` ustawia `Border` w tym samym obszarze, który jest widoczny w trybie normalnym.

Metoda obsługi zdarzenia `Loaded` sprawdza, czy dostępny jest obiekt `OrientationSensor`, i jeżeli jest, zaczyna pobieranie przez ustawienie właściwości `UriSource` w obiekcie `BitmapImage`. Po prawidłowym pobraniu obrazu odczytywane są jego wymiary, które razem z wymiarami strony są zapisywane w polach obiektu.

Listing 18.19. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    OrientationSensor orientationSensor = OrientationSensor.GetDefault();
    double pageWidth, pageHeight, maxDimension;
    int imageWidth, imageHeight;
    string title = "The Garden of Earthly Delights. Autor: Hieronymus Bosch";
    double zoomInScale;
    double rotation;
    bool isZoomView;

    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
        ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
        SizeChanged += OnMainPageSizeChanged;
    }
...
    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        if (orientationSensor == null)
        {
            await new MessageDialog("OrientationSensor nie jest dostępny",
                                    "Earthly Delights").ShowAsync();

            progressBar.Visibility = Visibility.Collapsed;
            statusText.Visibility = Visibility.Collapsed;
        }
        else
        {
            bitmapImage.UriSource =
                new Uri("http://upload.wikimedia.org/wikipedia/commons/6/62/
                ↪The_Garden_of_Earthly_Delights_by_Bosch_High_Resolution_2.jpg");
        }
    }

    void OnMainPageSizeChanged(object sender, SizeChangedEventArgs args)
    {
        // Zapisanie rozmiarów strony
        pageWidth = this.ActualWidth;
        pageHeight = this.ActualHeight;
        maxDimension = Math.Max(pageWidth, pageHeight);

        // Inicjalizacja wartości
```

```
        outlineBorder.Width = pageWidth;
        outlineBorder.Height = pageHeight;
        borderTransform.CenterX = pageWidth / 2;
        borderTransform.CenterY = pageHeight / 2;
    }

    void OnBitmapImageDownloadProgress(object sender, DownloadProgressEventArgs args)
    {
        progressBar.Value = args.Progress;
    }

    async void OnBitmapImageFailed(object sender, ExceptionRoutedEventArgs args)
    {
        progressBar.Visibility = Visibility.Collapsed;
        statusText.Visibility = Visibility.Collapsed;

        await new MessageDialog("Nie można pobrać obrazu: " + args.ErrorMessage,
                                "Earthly Delights").ShowAsync();
    }

    void OnBitmapImageOpened(object sender, RoutedEventArgs args)
    {
        progressBar.Visibility = Visibility.Collapsed;
        statusText.Visibility = Visibility.Collapsed;

        // Zapisanie wymiarów obrazu
        imageWidth = bitmapImage.PixelWidth;
        imageHeight = bitmapImage.PixelHeight;
        titleText.Text = String.Format("{0} ({1}\xOOD7{2})", title, imageWidth,
imageHeight);

        // Inicjalizacja przekształcenia obrazu
        zoomInScale = Math.Min(pageWidth / imageWidth, pageHeight / imageHeight);

        // Uruchomienie OrientationSensor
        if (orientationSensor != null)
        {
            ProcessNewOrientationReading(orientationSensor.GetCurrentReading());
            orientationSensor.ReportInterval = orientationSensor.MinimumReportInterval;
            orientationSensor.ReadingChanged += OnOrientationSensorReadingChanged;
        }
    }

    async void OnOrientationSensorReadingChanged(OrientationSensor sender,
                                                 OrientationSensorReadingChanged
                                                 ↪EventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                ProcessNewOrientationReading(args.Reading);
            });
    }
...
}
```

Metoda `ProcessNewOrientationReading` tworzy obiekt `Matrix3D` na podstawie `Sensor` `↪RotationMatrix` i korzysta z niego do uzyskania wartości `HorizontalCoordinate`.

Listing 18.20. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
void ProcessNewOrientationReading(OrientationSensorReading orientationReading)
{
    if (orientationReading == null)
        return;

    // Pobranie macierzy obrotu i konwersja na współrzędne horyzontalne
    SensorRotationMatrix m = orientationReading.RotationMatrix;

    if (m == null)
        return;

    Matrix3D matrix3d = new Matrix3D(m.M11, m.M12, m.M13, 0,
                                     m.M21, m.M22, m.M23, 0,
                                     m.M31, m.M32, m.M33, 0,
                                     0, 0, 0, 1);
    if (!matrix3d.HasInverse)
        return;

    HorizontalCoordinate horzCoord = HorizontalCoordinate.FromMotionMatrix(matrix3d);

    // Ustawienie środka przekształcenia w elemencie Image
    imageTransform.CenterX = (imageWidth + maxDimension) *
                             (180 + horzCoord.Azimuth) / 360 - maxDimension / 2;
    imageTransform.CenterY = (imageHeight + maxDimension) *
                             (90 - horzCoord.Altitude) / 180 - maxDimension / 2;

    // Ustawienie przesunięcia dla elementu Border
    borderTransform.TranslateX = imageTransform.CenterX - pageWidth / 2;
    borderTransform.TranslateY = imageTransform.CenterY - pageHeight / 2;

    // Pobranie obrotu z Tilt
    rotation = -horzCoord.Tilt;
    UpdateImageTransforms();
}
```

Metoda ta jest odpowiedzialna za ustawienie części transformacji; pozostałe są ustawiane w metodzie `UpdateImageTransforms` (której wywołanie następuje na końcu tej metody). Dla azymutu równego 0 (co następuje, gdy tablet jest skierowany na północ) oraz wysokości równej 0 (czyli gdy tablet jest ustawiony pionowo) właściwości `CenterX` i `CenterY` są ustawione na środek obrazu. W pozostałych przypadkach są ustawione na wartości obejmujące całą szerokość i wysokość, razem z marginesem, dzięki któremu będzie można wyświetlić obszar, w którym nie jest widoczny żaden fragment obrazu (w przeciwnym razie program musiałby pokazywać jednocześnie prawą krawędź obrazu na lewej stronie ekranu oraz lewą krawędź na prawej stronie ekranu).

Chciałem, aby operacja skalowania była animowana, więc dodałem do `MainPage` właściwość zależną sterującą animacją po stuknięciu ekranu.

Listing 18.21. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    // Właściwość zależna do animacji skalowania
    static readonly DependencyProperty interpolationFactorProperty =
        DependencyProperty.Register("InterpolationFactor",
```

```
                                    typeof(double),
                                    typeof(MainPage),
                                    new PropertyMetadata(0.0,
OnInterpolationFactorChanged));
...
    // Właściwość współczynnika interpolacji
    public static DependencyProperty InterpolationFactorProperty
    {
        get { return interpolationFactorProperty; }
    }

    public double InterpolationFactor
    {
        set { SetValue(InterpolationFactorProperty, value); }
        get { return (double)GetValue(InterpolationFactorProperty); }
    }
...
    protected override void OnTapped(TappedRoutedEventArgs e)
    {
        // Animacja właściwości InterpolationFactor
        DoubleAnimation doubleAnimation = new DoubleAnimation
        {
            EnableDependentAnimation = true,
            To = isZoomView ? 0 : 1,
            Duration = new Duration(TimeSpan.FromSeconds(1))
        };
        Storyboard.SetTarget(doubleAnimation, this);
        Storyboard.SetTargetProperty(doubleAnimation, "InterpolationFactor");
        Storyboard storyboard = new Storyboard();
        storyboard.Children.Add(doubleAnimation);
        storyboard.Begin();
        isZoomView ^= true;
        base.OnTapped(e);
    }

    static void OnInterpolationFactorChanged(DependencyObject obj,
                                             DependencyPropertyChangedEventArgs
                                             ↪args)
    {
        (obj as MainPage).UpdateImageTransforms();
    }
...
}
```

Metoda `OnInterpolationFactorChanged` wywołuje również `UpdateImageTransforms`, która realizuje większość trudnych zadań.

Listing 18.22. Projekt: EarthlyDelights | Plik: MainPage.xaml.cs (fragment)

```
void UpdateImageTransforms()
{
    // Jeżeli powiększony, ustaw skalowanie
    double interpolatedScale = 1 + InterpolationFactor * (zoomInScale - 1);
    imageTransform.ScaleX =
    imageTransform.ScaleY = interpolatedScale;

    // Przenieś środek przekształcenia na środek ekranu
```

```
    imageTransform.TranslateX = pageWidth / 2 - imageTransform.CenterX;
    imageTransform.TranslateY = pageHeight / 2 - imageTransform.CenterY;

    // Jeżeli pomniejszony, korekta dla skalowania
    imageTransform.TranslateX -= InterpolationFactor *
                        (pageWidth / 2 - zoomInScale * imageTransform.CenterX);
    imageTransform.TranslateY -= InterpolationFactor *
                        (pageHeight / 2 - zoomInScale * imageTransform.CenterY);

    // Jeżeli pomniejszony, wyśrodkuj obraz na ekranie
    imageTransform.TranslateX += InterpolationFactor *
                        (pageWidth - zoomInScale * imageWidth) / 2;
    imageTransform.TranslateY += InterpolationFactor *
                        (pageHeight - zoomInScale * imageHeight) / 2;

    // Ustaw grubość ramki
    outlineBorder.BorderThickness = new Thickness(2 / interpolatedScale);
    outlineRectangle.StrokeThickness = 2 / interpolatedScale;

    // Ustaw obrót i ramkę
    imageTransform.Rotation = (1 - InterpolationFactor) * rotation;
    borderTransform.Rotation = -rotation;
}
```

Metoda ta jest wywoływana, gdy pojawi się nowa wartość `OrientationSensor` lub gdy zmieni się wartość właściwości `InterpolationFactor` w czasie operacji skalowania. Jeżeli jesteś zainteresowany szczegółami działania tej metody, możesz ją uprościć przez wyeliminowanie całego kodu interpolacji. Ustaw `InterpolationFactor` na 0, a następnie na 1 — zobaczysz, że jest to dosyć proste.

Mapy oraz kafelki map Bing

Klasa `Geolocator` nie jest uznawana za sensor i jest zdefiniowana w całkowicie innej przestrzeni nazw — `Windows.Devices.Geolocation`. Jednak jest nieco podobna, ponieważ uruchamiamy ją, a następnie informuje nas ona o zmianie lokalizacji geograficznej.

Musimy jawnie wskazać w sekcji *Capabilities* pliku *Package.appxmanifest*, że nasza aplikacja wymaga uprawnienia *Location*. Przy pierwszym uruchomieniu aplikacji Windows 8 poprosi użytkownika o potwierdzenie.

Zazwyczaj korzystamy z lokalizacji zwracanej przez `Geolocator` w połączeniu z mapami. Kontrolka map Bing nie jest wbudowana w Windows 8, ale może być pobrana w postaci pakietu umożliwiającego dodanie jej do aplikacji. Konieczny jest również klucz uprawnień, który można uzyskać na witrynie *www.bingmapsportal.com*.

Jednak w ostatnim programie z tego rozdziału użyję nieco innego podejścia. Zamierzam pokazać mapę, która obraca się zgodnie z orientacją tabletu. Obrót taki pozwoli zorientować mapę względem rzeczywistej północy (lub innego odczytu zwracanego przez tablet). W tym celu nie będę korzystał z kontrolki Bing Maps. Zamiast tego użyję usługi SOAP Bing Maps, która pozwala na pobranie pojedynczych kafelków — połączę je w pełną mapę. Klucz uprawnień jest tu również wymagany.

Po uruchomieniu programu *RotatingMap* być może będziesz chciał przesuwać i skalować mapę palcami. To nie działa. Program nie posiada interfejsu dotykowego! Aby uprościć

przykład, sprawiłem, że program środkuje mapę w bieżącej lokalizacji i zmienia jej orientację po zmianie lokalizacji. Program posiada na pasku aplikacji przyciski do powiększania i zmniejszania skali oraz do przełączania się pomiędzy widokiem mapy a widokiem zdjęcia satelitarnego, ale to wszystko.

Zawartość pliku XAML zamieszczona jest na poniższym listingu. Wszystkie kafelki składające się na mapę są umieszczane w obiekcie `Canvas` o nazwie `imageCanvas`. Zwróć uwagę na element `RotateTransform` pozwalający obracać `Canvas` wokół jego środka.

Listing 18.23. Projekt: RotatingMap | Plik: MainPage.xaml (fragment)

```
<Page ... >

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Canvas Name="imageCanvas"
                HorizontalAlignment="Center"
                VerticalAlignment="Center">
            <Canvas.RenderTransform>
                <RotateTransform x:Name="imageCanvasRotate" />
            </Canvas.RenderTransform>
        </Canvas>

        <!-- Okrąg do pokazania lokalizacji -->
        <Ellipse Name="locationDisplay"
                 Width="24"
                 Height="24"
                 Stroke="Red"
                 StrokeThickness="6"
                 HorizontalAlignment="Center"
                 VerticalAlignment="Center"
                 Visibility="Collapsed" />

        <!-- Strzałka wskazująca północ -->
        <Border HorizontalAlignment="Left"
                VerticalAlignment="Top"
                Margin="12"
                Background="Black"
                Width="36"
                Height="36"
                CornerRadius="18">
            <Path Stroke="White"
                  StrokeThickness="3"
                  Data="M 18 4 L 18 24 M 12 12 L 18 4 24 12">
                <Path.RenderTransform>
                    <RotateTransform x:Name="northArrowRotate"
                                     CenterX="18"
                                     CenterY="18" />
                </Path.RenderTransform>
            </Path>
        </Border>

        <!-- Wyświetlenie "powered by bing" -->
        <Border Background="Black"
                HorizontalAlignment="Center"
                VerticalAlignment="Bottom"
                Margin="12"
                CornerRadius="12"
```

```
                Padding="3">

            <StackPanel Name="poweredByDisplay"
                        Orientation="Horizontal"
                        Visibility="Collapsed">
                <TextBlock Text=" powered by "
                           Foreground="White"
                           VerticalAlignment="Center" />
                <Image Stretch="None">
                    <Image.Source>
                        <BitmapImage x:Name="poweredByBitmap" />
                    </Image.Source>
                </Image>
            </StackPanel>
        </Border>
    </Grid>

    <Page.BottomAppBar>
        <AppBar Name="bottomAppBar"
                IsEnabled="False">
            <StackPanel Orientation="Horizontal"
                        HorizontalAlignment="Right">

                <!-- Usunięcie odwołania do BackgroundCheckedGlyph w
                     AppBarButtonStyle w celu użycia go jako CheckBox -->

                <CheckBox Name="streetViewAppBarButton"
                          Style="{StaticResource StreetAppBarButtonStyle}"
                          AutomationProperties.Name="Street View"
                          Checked="OnStreetViewAppBarButtonChecked"
                          Unchecked="OnStreetViewAppBarButtonChecked" />

                <Button Name="zoomInAppBarButton"
                        Style="{StaticResource ZoomInAppBarButtonStyle}"
                        Click="OnZoomInAppBarButtonClick" />

                <Button Name="zoomOutAppBarButton"
                        Style="{StaticResource ZoomOutAppBarButtonStyle}"
                        Click="OnZoomOutAppBarButtonClick" />
            </StackPanel>
        </AppBar>
    </Page.BottomAppBar>
</Page>
```

Możliwe jest „ręczne” wywoływanie usług SOAP Bing Maps przez przesyłanie i analizowanie plików XML, ale znacznie bezpieczniejszym podejściem jest korzystanie z usługi sieciowej poprzez klasy pośredniczące generowane przez Visual Studio. Klasy pośredniczące powodują, że usługa sieciowa jest reprezentowana przez zestaw struktur, typów wyliczeniowych i asynchronicznych wywołań metod. Aby dodać te klasy do programu *RotatingMap*, kliknąłem prawym przyciskiem myszy nazwę projektu w oknie *Solution Explorer* i wybrałem z menu opcję *Add Service Reference*. Gdy zostało wyświetlone okno dialogowe z adresem, wkleiłem do niego URL Imagery Service (który możesz znaleźć na stronie *http://msdn.microsoft.com/en-us/library/cc966738.aspx* zawierającej adresy trzech innych usług sieciowych połączonych z Bing Maps). Nadałem jej nazwę *ImageryService*, co spowodowało, że Visual Studio wygenerował kod w przestrzeni nazw `RotatingMap.ImageryService`.

Usługa zawiera dwa typy żądań — `GetMapUriAsync` oraz `GetImageryMetadataAsync`. Pierwszy typ pozwala na uzyskanie statycznej mapy dla określonej lokalizacji, ale ja skorzystałem z drugiego, który pozwala na pozyskanie informacji potrzebnych do pobrania kafelków mapy, które razem tworzą kompletną mapę.

Zacznijmy zapoznawanie się z kodem *RotatingMap* od konstruktora klasy `MainPage`. Jak widać, zapisuje on w ustawieniach aplikacji tylko dwie wartości — styl mapy (wartość typu wyliczeniowego `MapStyle`, który jest generowany w czasie dodawania usługi sieciowej do projektu zawierającego wartości określające mapę lub zdjęcia satelitarne) oraz całkowity poziom powiększenia.

Listing 18.24. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    // Zapisanie jako ustawienia aplikacji
    MapStyle mapStyle = MapStyle.Aerial;
    int zoomLevel = 12;
    public MainPage()
    {
        this.InitializeComponent();
        DisplayProperties.AutoRotationPreferences =
        ↪DisplayProperties.NativeOrientation;
        Loaded += OnMainPageLoaded;
        SizeChanged += OnMainPageSizeChanged;

        // Pobranie ustawień aplikacji (i ich późniejsze zapisanie)
        IPropertySet propertySet = ApplicationData.Current.LocalSettings.Values;

        if (propertySet.ContainsKey("ZoomLevel"))
            zoomLevel = (int)propertySet["ZoomLevel"];

        if (propertySet.ContainsKey("MapStyle"))
            mapStyle = (MapStyle)(int)propertySet["MapStyle"];

        Application.Current.Suspending += (sender, args) =>
        {
            propertySet["ZoomLevel"] = zoomLevel;
            propertySet["MapStyle"] = (int)mapStyle;
        };
    }
...
}
```

Usługa sieciowa jest wykorzystywana wyłącznie w metodzie obsługi zdarzenia `Loaded`. Konieczne jest wykonanie dwóch wywołań — jednego do pobrania metadanych mapy dla widoku mapy oraz drugiego dla widoku satelitarnego. Dane te są zapisywane w dwóch obiektach lokalnej klasy o nazwie `ViewParams`. Najważniejszą częścią metadanych jest szablon URI do pobierania poszczególnych kafelków mapy. Klasa `ViewParams` posiada również pola na maksymalny i minimalny poziom powiększenia, ale wiem, że zmienia się on w zakresie od 1 do 21 i w kolejnych częściach kodu zakładam maksymalny poziom równy 21.

Listing 18.25. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    // Parametry przechowywania dla dwóch widoków
    class ViewParams
    {
        public string UriTemplate;
        public int MinimumLevel;
        public int MaximumLevel;
    }
    ViewParams aerialParams;
    ViewParams roadParams;

    Geolocator geolocator = new Geolocator();
    Inclinometer inclinometer = Inclinometer.GetDefault();
...
    async void OnMainPageLoaded(object sender, RoutedEventArgs args)
    {
        // Inicjowanie usługi obrazów Bing Maps
        ImageryServiceClient imageryServiceClient =
            new ImageryServiceClient(
                ImageryServiceClient.EndpointConfiguration.
                ➥BasicHttpBinding_IImageryService);

        // Wykonanie dwóch żądań dla mapy i zdjęć
        ImageryMetadataRequest request = new ImageryMetadataRequest
        {
            Credentials = new Credentials
            {
                ApplicationId = "AkNpobMGtsXUh9o8T9j1doUmjhtcoexUHHTClyBILqnrag-
                ➥ibtcHruZQPFo61cn8"
            },
            Style = MapStyle.Road
        };
        Task<ImageryMetadataResponse> roadStyleTask =
                    imageryServiceClient.GetImageryMetadataAsync(request);

        request = new ImageryMetadataRequest
        {
            Credentials = new Credentials
            {
                ApplicationId = "Tu wstaw klucz uprawnień Bing"
            },
            Style = MapStyle.Aerial
        };
        Task<ImageryMetadataResponse> aerialStyleTask =
                    imageryServiceClient.GetImageryMetadataAsync(request);

        // Oczekiwanie na zakończenie obu zadań
        Task.WaitAll(roadStyleTask, aerialStyleTask);

        // Sprawdzenie poprawności
        if (!roadStyleTask.IsCanceled && !roadStyleTask.IsFaulted &&
            !aerialStyleTask.IsCanceled && !aerialStyleTask.IsCanceled)
        {
```

```
            // Pobranie mapy bitowej "powered by"
            poweredByBitmap.UriSource = roadStyleTask.Result.BrandLogoUri;
            poweredByDisplay.Visibility = Visibility.Visible;

            // Pobranie URI oraz minimalnych i maksymalnych poziomów powiększenia
            roadParams = CreateViewParams(roadStyleTask.Result.Results[0]);
            aerialParams = CreateViewParams(aerialStyleTask.Result.Results[0]);

            // Pobranie bieżącej lokalizacji
            Geoposition geoPosition = await geolocator.GetGeopositionAsync();
            GetLongitudeAndLatitude(geoPosition.Coordinate);
            RefreshDisplay();

            // Pobranie zaktualizowanej lokalizacji
            geolocator.PositionChanged += OnGeolocatorPositionChanged;

            // Aktywowanie paska aplikacji
            bottomAppBar.IsEnabled = true;
            streetViewAppBarButton.IsChecked = mapStyle == MapStyle.Road;

            // Pobranie bieżącego obrotu
            if (inclinometer != null)
            {
                SetRotation(inclinometer.GetCurrentReading());
                inclinometer.ReadingChanged += OnInclinometerReadingChanged;
            }
        }
    }

    ViewParams CreateViewParams(ImageryMetadataResult result)
    {
        string uri = result.ImageUri;
        uri = uri.Replace("{subdomain}", result.ImageUriSubdomains[0]);
        uri = uri.Replace("&token={token}", "");
        uri = uri.Replace("{culture}", "en-us");

        return new ViewParams
        {
            UriTemplate = uri,
            MinimumLevel = result.ZoomRange.From,
            MaximumLevel = result.ZoomRange.To
        };
    }
...
}
```

Do odczytania metadanych dla dwóch widoków potrzebne są dwa wywołania asynchroniczne, ale ponieważ nie są one od siebie zależne, mogą być wykonywane w tym samym czasie. Jest to świetne zastosowanie dla metody `Task.WaitAll`, która czeka na zakończenie wykonywania wielu obiektów `Task`.

Gdy oba wywołania usługi sieciowej zakończą się prawidłowo, tworzone są obiekty `Geolocator` oraz `Inclinometer`. Obiekt `Inclinometer` jest używany wyłącznie do odczytania wartości odchylenia używanej przy obracaniu mapy oraz do obracania strzałki wskazującej północ.

Listing 18.26. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    async void OnInclinometerReadingChanged(Inclinometer sender,
                                            InclinometerReadingChangedEventArgs args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            SetRotation(args.Reading);
        });
    }

    void SetRotation(InclinometerReading inclinometerReading)
    {
        if (inclinometerReading == null)
            return;

        imageCanvasRotate.Angle = inclinometerReading.YawDegrees;
        northArrowRotate.Angle = inclinometerReading.YawDegrees;
    }
...
}
```

Na tym kończy się metoda obsługi zdarzenia `Loaded` i program posiada teraz dwa szablony adresu URI, za pomocą których może pobrać poszczególne kafelki mapy. Kafelki tworzące podstawową mapę Bing są kwadratowymi mapami bitowymi o boku 256 pikseli. Każdy kafelek jest skojarzony z określoną długością i szerokością geograficzną, a także poziomem powiększenia oraz zawiera obraz fragmentu świata spłaszczony za pomocą odwzorowania walcowego równokątnego.

Na poziomie 1. cała Ziemia — a właściwie część Ziemi o szerokościach geograficznych pomiędzy dodatnim i ujemnym stopniem 85,05 — jest odwzorowana za pomocą czterech kafelków:

Wkrótce przedstawię sens liczb w kafelkach. Kafelki mają bok o 256 pikselach, więc na równiku każdy piksel obejmuje około 78 kilometrów.

Na poziomie 2. Ziemia jest odwzorowana przez 16 kafelków.

Kafelki mają bok o 256 pikselach, więc na równiku każdy piksel obejmuje około 38 kilometrów.

Każdy kafelek z poziomu 1. obejmuje taki sam obszar jak cztery kafelki z poziomu 2. i trend jest kontynuowany — poziom 3. zawiera 64 kafelki, poziom 4. zawiera 256 kafelków, i tak aż do poziomu 21., który (zgodnie z zasadą) odwzorowuje całą Ziemię za pomocą ponad 4 trylionów kafelków — 2 miliony poziomo i 2 miliony pionowo, przy rozdzielczości na równiku równej 7,5 cm na piksel.

W jaki sposób można spójnie zorganizować tak wiele kafelków? Pamiętaj, że użyte są tu trzy wymiary — długość, szerokość i poziom powiększenia — a dla maksymalnej efektywności przesyłania tych kafelków poprzez usługę sieciową kafelki obejmujące ten sam obszar powinny być przechowywane na serwerach obok siebie.

Oczywiście został użyty bardzo sprytny sposób numerowania, nazywany *quadkey*. Każdy kafelek ma unikatową wartość `quadkey`. Szablony URI odczytane z usługi Bing Maps zawierają znacznik `{quadkey}`, który powinieneś zastąpić odwołaniem do odpowiedniego kafelka. Przedstawione wcześniej rysunki mają pokazane wartości `quadkey` w lewym górnym narożniku każdego kafelka. Wiodące zera są istotne! Liczba cyfr w numerze jest równa poziomowi powiększenia. Kafelki na poziomie 21. są identyfikowane za pomocą 21 znakowych wartości `quadkey`.

W kodzie tym używane są wyłącznie cyfry 0, 1, 2 i 3, czyli są to liczby w systemie czwórkowym. Binarnie zapisane liczby 0, 1, 2 i 3 to 00, 01, 10 i 11. Pierwszy bit jest współrzędną pionową, a drugi współrzędną poziomą. Dzięki temu bity odpowiadają przeplatanej długości i szerokości.

Jak już wiesz, każdy kafelek na poziomie 1. odpowiada czterem kafelkom z poziomu 2., więc można uważać, że kafelki mają relacje „nadrzędny – podrzędny”. Wartość `quadkey` kafelka podrzędnego zaczyna się tymi samymi cyframi co wartość jego kafelka nadrzędnego, ale dodana jest jeszcze jedna cyfra określająca lokalizację w tym elemencie nadrzędnym.

Można łatwo pobrać wartość `quadkey` kafelka nadrzędnego z podrzędnego, odcinając po prostu ostatnią cyfrę.

Aby użyć usługi sieciowe Bing Maps, konieczne jest określenie wartości `quadkey` dla dowolnej długości i szerokości geograficznej. Kod pokazanej w następnym listingu metody `GetLongitudeAndLatitude` ilustruje pierwszy krok, czyli konwersję długości i szerokości geograficznej z `Geolocator` na wartości `double` z zakresu od 0 do 1, a następnie na wartości całkowite:

Listing 18.27. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
    const int BITRES = 29;
...
    int integerLongitude = -1;
    int integerLatitude = -1;
...
    async void OnGeolocatorPositionChanged(Geolocator sender, PositionChangedEventArgs
    ↪args)
    {
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                GetLongitudeAndLatitude(args.Position.Coordinate);
                RefreshDisplay();
            });
    }

    void GetLongitudeAndLatitude(Geocoordinate geoCoordinate)
    {
        locationDisplay.Visibility = Visibility.Visible;

        // Obliczenie całkowitej długości i szerokości geograficznej
        double relativeLongitude = (180 + geoCoordinate.Longitude) / 360;
        integerLongitude = (int)(relativeLongitude * (1 << BITRES));

        double sinTerm = Math.Sin(Math.PI * geoCoordinate.Latitude / 180);
        double relativeLatitude = 0.5 - Math.Log((1 + sinTerm) / (1 - sinTerm)) / (4 *
        ↪Math.PI);
        integerLatitude = (int)(relativeLatitude * (1 << BITRES));
    }
...
}
```

Wartość BITRES wynosi 29, co obejmuje 21 bitów wartości quadkey na poziomie 21. oraz 8 bitów na wielkość kafelka w pikselach, co oznacza, że te wartości całkowite identyfikują długość i szerokość geograficzną z dokładnością do najbliższego piksela kafelka na najwyższym poziomie powiększenia. Wyliczenie wartości `integerLongitude` jest bardzo proste, ale `integerLatitude` znacznie bardziej skomplikowane, ponieważ przekształcenie mapy typu Mercator powoduje kompresję szerokości geograficznych wraz ze wzrostem odległości od równika.

Na przykład środek Central Parku w Nowym Jorku ma długość geograficzną równą –73,965368 oraz szerokość 40,783271. Względne wartości `double` (zaokrąglone do kilku miejsc po przecinku) wynoszą 0,29454 i 0,37572. 29-bitowe wartości całkowite (pokazane w systemie binarnym i pogrupowane po cztery cyfry dla zwiększenia czytelności) wynoszą:

```
0 1001 0110 1100 1110 0000 1000 0000
0 1100 0000 0101 1110 1011 0000 0000
```

Załóżmy, że potrzebujesz kafelka dla tej długości i szerokości geograficznej i powiększenia poziomu 12. Musimy użyć górnych 12 bitów tej liczby określającej długość i szerokość geograficzną (uważaj — wynikowe cyfry są grupowane nieco inaczej).

```
0100 1011 01100110 0000 0010
```

Są to dwie liczby binarne, ale w celu utworzenia wartości quadkey muszą być one być połączone, aby utworzyły liczbę o podstawie 4. Nie można zrobić tego w kodzie bez przeglądania bitów w pętli, ale dla celów demonstracji możesz po prostu pomnożyć przez dwa wszystkie bity szerokości i dodać te dwie wartości tak, jakby były wartościami o podstawie 4:

```
  0100 1011 0110
+ 0220 0000 0020
  --------------
  0320 1011 0130
```

Jest to wartość, którą należy podstawić w miejsce symbolu {quadkey} w szablonie URI pobranym z usługi sieciowej. Wynikowy URI definiuje kwadratową mapę bitową o boku 256 pikseli.

Poniżej przedstawiony jest kod z *RotatingMap*, w którym na podstawie długości i szerokości geograficznej tworzona jest wartość quadkey. Dla zwiększenia czytelności kod został rozdzielony, aby pokazać na początku wyznaczanie wartości całkowitej, a następnie ciągu znaków:

Listing 18.28. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    StringBuilder strBuilder = new StringBuilder()
...
    string ToQuadKey(int longitude, int latitude, int level)
    {
        long quadkey = 0;
        int mask = 1 << (level - 1);

        for (int i = 0; i < level; i++)
        {
            quadkey <<= 2;

            if ((longitude & mask) != 0)
                quadkey |= 1;

            if ((latitude & mask) != 0)
                quadkey |= 2;

            mask >>= 1;
        }

        strBuilder.Clear();

        for (int i = 0; i < level; i++)
```

```
        {
            strBuilder.Insert(0, (quadkey & 3).ToString());
            quadkey >>= 2;
        }

        return strBuilder.ToString();
    }
...
}
```

Wartość quadkey określa kafelek zawierający oczekiwaną długość i szerokość geograficzną i nasza aktualna lokalizacja znajduje się gdzieś wewnątrz tego kafelka. Lokalizacja piksela w kafelku może być definiowana przez następne 8 cyfr w długości i szerokości geograficznej wymaganej do wyliczenia quadkey.

Mamy już prawie wszystkie elementy układanki. Ponieważ cała strona musi być zapełniona kwadratowymi kafelkami o boku 256 pikseli, ta tablica kafelków musi dać się obracać; bieżąca pozycja użytkownika znajduje się na środku ekranu, zatem metoda obsługi zdarzenia `SizeChanged` wylicza liczbę potrzebnych kafelków, więc jednocześnie liczbę potrzebnych elementów `Image`. Pole o nazwie `sqrtNumTiles` oznacza „pierwiastek z liczby kafelków". Dla ekranu 1366 na 768 pikseli wynosi on 9. Całkowita liczba kafelków (i elementów `Image`) jest kwadratem tej liczby, czyli 81.

Listing 18.29. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page
{
...
    int sqrtNumTiles;        // Zawsze liczba nieparzysta
...
    void OnMainPageSizeChanged(object sender, SizeChangedEventArgs args)
    {
        // Usuwanie istniejących elementów Image
        imageCanvas.Children.Clear();

        // Określenie liczby potrzebnych elementów
        double diagonal = Math.Sqrt(Math.Pow(args.NewSize.Width, 2) +
                                    Math.Pow(args.NewSize.Height, 2));

        sqrtNumTiles = 1 + 2 * (int)Math.Ceiling((diagonal / 2) / 256);

        // Tworzenie elementów Image dla tablicy sqrtNumTiles * sqrtNumTiles
        for (int i = 0; i < sqrtNumTiles * sqrtNumTiles; i++)
        {
            Image image = new Image
            {
                Source = new BitmapImage(),
                Stretch = Stretch.None
            };
            imageCanvas.Children.Add(image);
        }
        RefreshDisplay();
    }
...
}
```

Metoda `RefreshDisplay` realizuje główną część zadania, przegląda w pętli elementy `Image` i określa ich wartości quadkey (więc i jednocześnie URI).

Listing 18.30. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page{
...
    void RefreshDisplay()
    {
        if (roadParams == null || aerialParams == null)
            return;

        if (integerLongitude == -1 || integerLatitude == -1)
            return;

        // Pobranie współrzędnych i przesunięcia pikseli na podstawie bieżącego powiększenia
        int croppedLongitude = integerLongitude >> BITRES - zoomLevel;
        int croppedLatitude = integerLatitude >> BITRES - zoomLevel;
        int xPixelOffset = (integerLongitude >> BITRES - zoomLevel - 8) % 256;
        int yPixelOffset = (integerLatitude >> BITRES - zoomLevel - 8) % 256;

        // Przygotowanie pętli
        string uriTemplate = (mapStyle == MapStyle.Road ? roadParams :
        ↪aerialParams).UriTemplate;
        int index = 0;
        int maxValue = (1 << zoomLevel) - 1;

        // Pętla po tablicy elementów Image
        for (int row = -sqrtNumTiles / 2; row <= sqrtNumTiles / 2; row++)
            for (int col = -sqrtNumTiles / 2; col <= sqrtNumTiles / 2; col++)
            {
                // Tworzenie Image i BitmapImage
                Image image = imageCanvas.Children[index] as Image;
                BitmapImage bitmap = image.Source as BitmapImage;
                index++;

                // Sprawdzenie, czy wykroczyliśmy poza granice
                if (croppedLongitude + col < 0 ||
                    croppedLongitude + col > maxValue ||
                    croppedLatitude + row < 0 ||
                    croppedLatitude + row > maxValue)
                {
                    bitmap.UriSource = null;
                }
                else
                {
                    // Obliczenie quadkey i ustawienie URI mapy bitowej
                    int longitude = croppedLongitude + col;
                    int latitude = croppedLatitude + row;
                    string strQuadkey = ToQuadKey(longitude, latitude, zoomLevel);
                    string uri = uriTemplate.Replace("{quadkey}", strQuadkey);
                    bitmap.UriSource = new Uri(uri);
                }

                // Pozycjonowanie elementu Image
                Canvas.SetLeft(image, col * 256 - xPixelOffset);
                Canvas.SetTop(image, row * 256 - yPixelOffset);
```

```
            }
    }
...
}
```

Pozostało tylko obsłużyć przyciski paska aplikacji. Przyciski powiększenia i pomniejszenia są włączane i wyłączane na podstawie minimalnych i maksymalnych poziomów powiększenia bieżącego widoku, choć w innych częściach programu przyjąłem maksymalną wartość powiększenia równą 21.

Listing 18.31. Projekt: RotatingMap | Plik: MainPage.xaml.cs (fragment)

```
public sealed partial class MainPage : Page{
...
    void OnStreetViewAppBarButtonChecked(object sender, RoutedEventArgs args)
    {
        ToggleButton btn = sender as ToggleButton;
        ViewParams viewParams = null;

        if (btn.IsChecked.Value)
        {
            mapStyle = MapStyle.Road;
            viewParams = roadParams;
        }
        else
        {
            mapStyle = MapStyle.Aerial;
            viewParams = aerialParams;
        }

        zoomLevel = Math.Max(viewParams.MinimumLevel,
                    Math.Min(viewParams.MaximumLevel, zoomLevel));

        RefreshDisplay();
        RefreshButtons();
    }

    void OnZoomInAppBarButtonClick(object sender, RoutedEventArgs args)
    {
        zoomLevel += 1;
        RefreshDisplay();
        RefreshButtons();
    }

    void OnZoomOutAppBarButtonClick(object sender, RoutedEventArgs args)
    {
        zoomLevel -= 1;
        RefreshDisplay();
        RefreshButtons();
    }

    void RefreshButtons()
    {
        ViewParams viewParams = streetViewAppBarButton.IsChecked.Value ? roadParams :
        ↪aerialParams;
        zoomInAppBarButton.IsEnabled = zoomLevel < viewParams.MaximumLevel;
```

```
            zoomOutAppBarButton.IsEnabled = zoomLevel > viewParams.MinimumLevel;
        }
    }
```

Nie jesteśmy przyzwyczajeni do oglądania znajomych regionów na obróconej mapie, więc w tym widoku wyspa Manhattan wygląda nieco dziwnie:

Jeżeli jesteś jednak w nieznanym miejscu, trzymając w ręku tablet i próbując określić miejsce, gdzie się znajdujesz, to orientacja mapy względem rzeczywistych warunków może być bardzo pomocna. Może kiedyś etykiety nazw miast i ulic również będą się dały obracać.

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ź

Ż

Twórz najlepsze aplikacje dla systemu Windows 8®

Sztandarowy produkt giganta z Redmond — Windows 8 — na dobre zagościł na rynku komputerowym. Już po pierwszym uruchomieniu rzuca się w oczy nowy interfejs użytkownika. Kafelki, bo o nich mowa, wzbudziły skrajne emocje. Część użytkowników wyjątkowo polubiła nowy wygląd ekranu, a inni mają problemy z przyzwyczajeniem się do niego. Jedno jest pewne — twórcy aplikacji powinni podążać za trendami wyznaczonymi przez Windows 8 i tworzyć spójne, przejrzyste aplikacje, wykorzystujące potencjał platformy.

Właśnie trzymasz w rękach najlepszą książkę poświęconą programowaniu dla platformy Windows. Jej najnowsze wydanie zostało zaktualizowane o wszystkie nowości, które pojawiły się w Windows 8. W trakcie lektury dowiesz się, co to jest XAML, poznasz jego składnię oraz możliwości. W kolejnych rozdziałach nauczysz się obsługiwać zdarzenia, tworzyć animacje, korzystać z wywołań asynchronicznych oraz wykonywać operacje natywne. Druga część książki zawiera informacje o funkcjach specjalnych: obsłudze urządzeń dotykowych, kamer, drukarek i rysików. Książka ta jest kompletnym przewodnikiem dla wszystkich programistów C# i osób zainteresowanych tym językiem.

Sięgnij po tę książkę i:

- poznaj nowości systemu Windows 8
- zobacz, jak tworzyć interfejs użytkownika Metro
- projektuj aplikacje na ekrany dotykowe
- odnieś sukces na rynku aplikacji dla platformy Windows 8

Microsoft

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

ISBN 978-83-246-7383-4
9 788324 673834
Cena 129,00 zł