Creating a reflected image with QML

This article shows how to create a **nice reflection effect for Image elements in QML**.

Introduction

The standard QML Image element allows to display a locally or remotely stored image: by creating a new Component, this article shows how to build a reusable element that automatically adds a reflection effect to a standard Image element, with those additional features:

- choose the color of the reflection fade effect
- choose the portion (ratio) of the original image to be reflected

Building the component

First, an empty **ReflectedImage.qml** file is created, to contain the new component. A Rectangle is used as root element, defining three properties to hold the relevant component information.

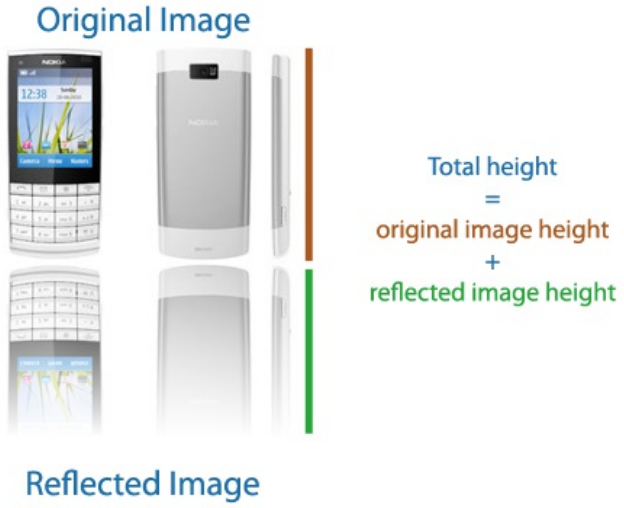
```
Rectangle {
    id: imageRoot

    // color applied to the reflection fade
    property string fadeColor : "#000000"

    // the image source
    property string source

    // ratio of the original image to be reflected
    property real reflectRatio : 0.5
}
```

Please Note: the ReflectedImage component assumes that the fadeColor effect is defined by using a color in the "#RRGGBB" form. So, both color keywords (for instance, "red" or "blue") and colors in the "#AARRGGBB" form will not work.

Defining the component size

The component is built by using **two separate Image elements**: one to display the original image, another to display the reflected image. As the picture below shows, the total height of the component will be determined by the sum of the original and reflected images.

(C) Copyright Nokia 2013. All rights reserved.




```
Rectangle {
    id: imageRoot

    [...]

    width: originalImage.width
    height: originalImage.height * (1 + reflectRatio)
}
```

Component's clipping

Since the reflectRatio property defines which portion of the original image should be reflected, the remaining part should be accordingly hidden. This can be easily accomplished by using the Item's clip property, that hides the content that overflows the Item's boundaries:

```
Rectangle {
    id: imageRoot

    [...]

    clip: true
}
```

Showing the reflected image

It's time to add the Image elements to the ReflectedImage component. As shown above, two different images are needed, placed one under the other. Both images must display the same image, whose source is defined by the component's source property:

```
Rectangle {
    id: imageRoot

    [...]

    Image {
 id: originalImage

 source: imageRoot.source
    }

    Image {
 id: reflectedImage

 anchors.top: originalImage.bottom
 source: imageRoot.source
    }
}
```

Now, the reflectedImage Image element must be flipped vertically, to correctly show the reflection effect. This is accomplished by performing a rotation of the element around the x axis, using a Rotation element:

```
Image {
    id: reflectedImage

    anchors.top: originalImage.bottom
    source: imageRoot.source

    transform: Rotation {
  origin.x: reflectedImage.width / 2
  origin.y: reflectedImage.height / 2
  axis.x: 1; axis.y: 0; axis.z: 0
  angle: 180
     }
}
```

Adding the fade effect

The last thing to add is the fade effect of the reflected image. To do this, a Rectangle with an applied Gradient is shown above the reflected image: the gradient will start with a transparent color (so that the reflected image is visible) and ends with a solid color (so that the reflected image gradually

(C) Copyright Nokia 2013. All rights reserved.

disappears).

```
Rectangle {
    id: imageRoot

    [...]

    Image {
 id: originalImage

 [...]
    }

    Image {
 id: reflectedImage

 [...]
    }

    Rectangle {
 anchors.top: originalImage.bottom
 width:  originalImage.width
 height: originalImage.height * reflectRatio

 gradient: Gradient {

     GradientStop {
  position: 0.0
  color: "#00" + fadeColor.substring(1)
     }
     GradientStop {
  position: 1.0
  color: "#FF" + fadeColor.substring(1)
     }
 }
    }
}
```

Using the component

Using the ReflectedImage component is straightforward: just like a standard Image element, it is just necessary to set its "source" property. Optionally, the "fadeColor" and "reflectRatio" properties can be defined as well:

```
ReflectedImage {
    anchors.centerIn: parent
    fadeColor: "#FFFFFF"
    reflectRatio: 0.9
    source: "pic2.jpg"
}
```

Related Content

A Qt Creator project containing the code presented in this article is available here: File:QMLReflectedImage.zip

A similar implementation was found on the Qt Bug Tracker website.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.