

Графика в QML

Язык QML предоставляет возможности задания цветов, шрифтов, манипулирования растровыми изображениями, создания градиентов и рисования графических примитивов на элементах холста.



1. Цвета

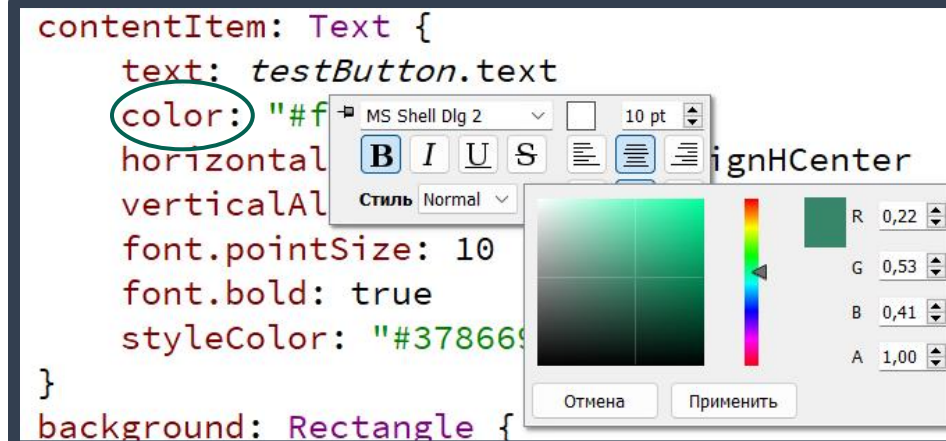
Цвета в QML можно задавать в виде *строк* или использовать *встроенную функцию Qt*.
Строковое задание цвета осуществляется **по имени** либо в **формате числового кода**

Строки имен - это стандарт, используемый в SVG.
Например: "red", "green", "darkkhaki", "snow"

Строки числовых кодов - цвета задаются в следующем формате: **#rrggbb**, принятом в HTML:
"#FF0000", "#00FE00", "#0000AF"

Встроенная функция **rgba()** позволяет получать различные цветовые значения. Ее использование выглядит следующим образом:

```
Rectangle {  
    color: Qt.rgb(0.3, 0.45, 0.21)  
    opacity: 0.5  
}
```



*Qt Creator дает в помощь соответствующий интерактивный инструмент. Для того чтобы им воспользоваться, встаньте в коде на позицию свойства **color**, выполните вызов контекстного меню и в этом меню выберите пункт «Показать панель Qt Quick»

2. Image

Чтобы файлы растровых и векторных изображений можно было использовать в языке QML, они должны быть в формате JPG, PNG или SVG.

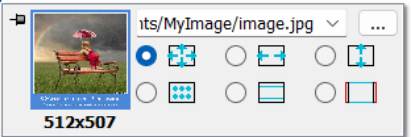
Элемент **Image** отображает файл изображения, указанный в свойстве **source**. Этот файл может находиться как на локальном диске компьютера, так и в сети. Элемент **Image** поддерживает не только растровые, но и векторные изображения в формате **SVG**.

Для трансформации изображения используются свойства **scale**, **rotation**. Эти трансформации по умолчанию осуществляются относительно центральной точки самого элемента. Для изменения точки для трансформации и поворота нужно задать соответствующее значение свойства **transformOrigin**.

```
Image {  
    ...  
    transformOrigin: Item.Top  
    ...  
}
```

Поворот
изображения
относительно
верха элемента

```
Image {  
    id: img  
    x: 0  
    y: 0  
    smooth: true  
    source: "c:/Users/Admin/Documents/MyImage/image.jpg"  
    scale: 0.75  
    rotation: -0.0  
}
```

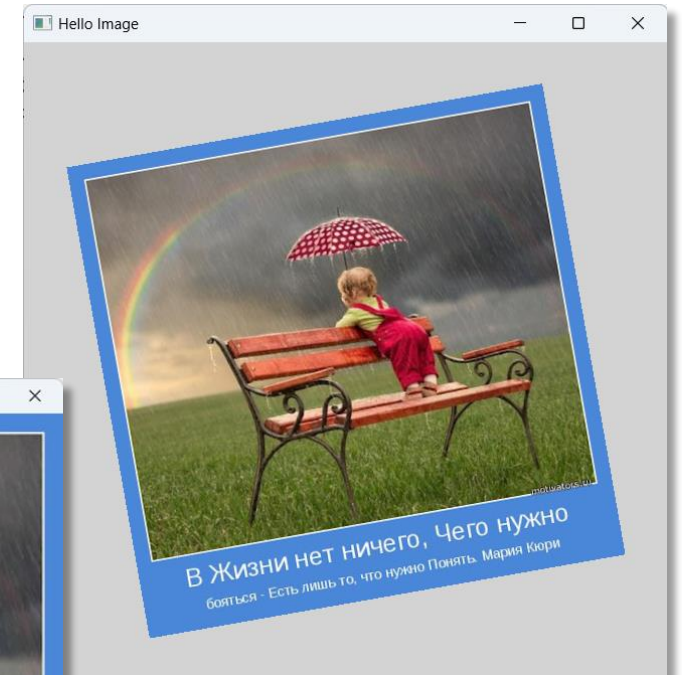
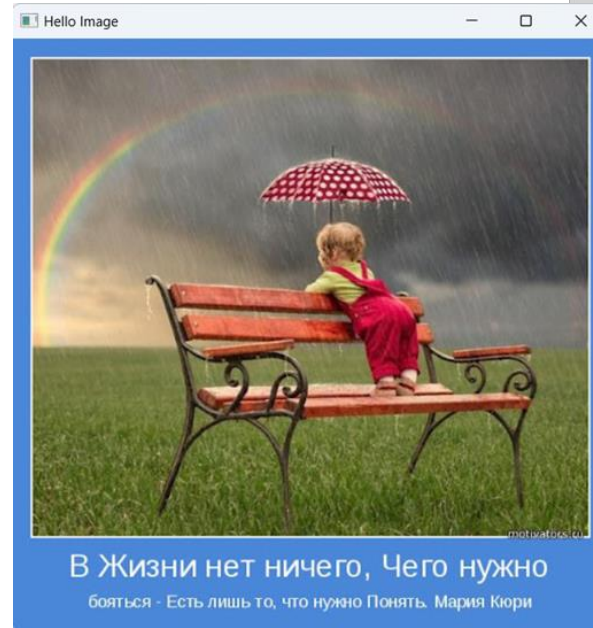


*Для настройки свойств элемента **Image** в Qt Creator так же имеется интерактивный редактор. Чтобы его открыть, встаньте на сам элемент **Image** и вызовите контекстное меню, в котором выберите пункт «Показать панель Qt Quick».

3. Отображение графического файла

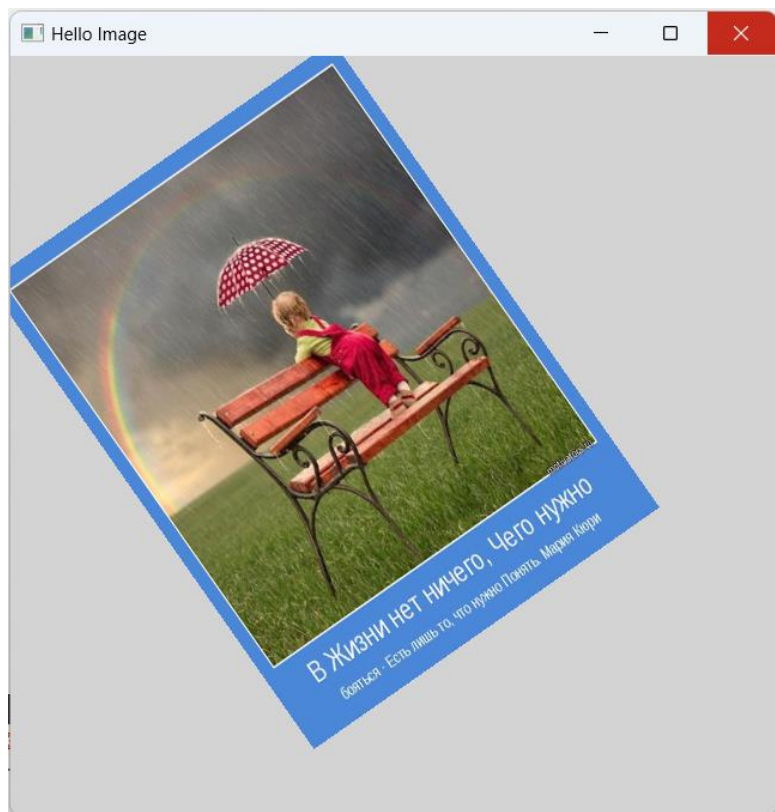
```
import QtQuick 2.15
Rectangle {
    color: "lightgray"
    width: img.width
    height: img.height
    Image {
        id: img
        x: 0 ; y: 0
        smooth: true
        source: "qrc:/image.jpg"
        scale: 1.0 //0.75
        rotation: 0.0 //-10.0
    }
}
```

Инициализация width и height происходит неявно при загрузке, и соответствуют размеру изображения, находящегося в файле.



4. Трансформации графического файла

Если трансформаций больше 1 они должны указываться в виде списка [... , ...]



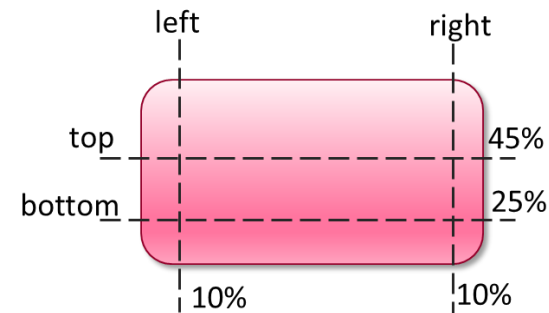
Середина изображения до трансформации

Для более тонкой настройки трансформации ее можно задавать при помощи элементов.

```
import QtQuick 2.15
Image {
    id: img
    x: 0 ; y: 0
    smooth: true
    source: "qrc : / image.jpg"
    transform: [
        Scale {
            origin.x: width / 3
            origin.y: height / 2
            xScale: 0.55
            yScale: 0.75
        },
        Rotation {
            origin.x: width / 3
            origin.y: height / 2
            angle: -35.0
        }
    ]
}
```

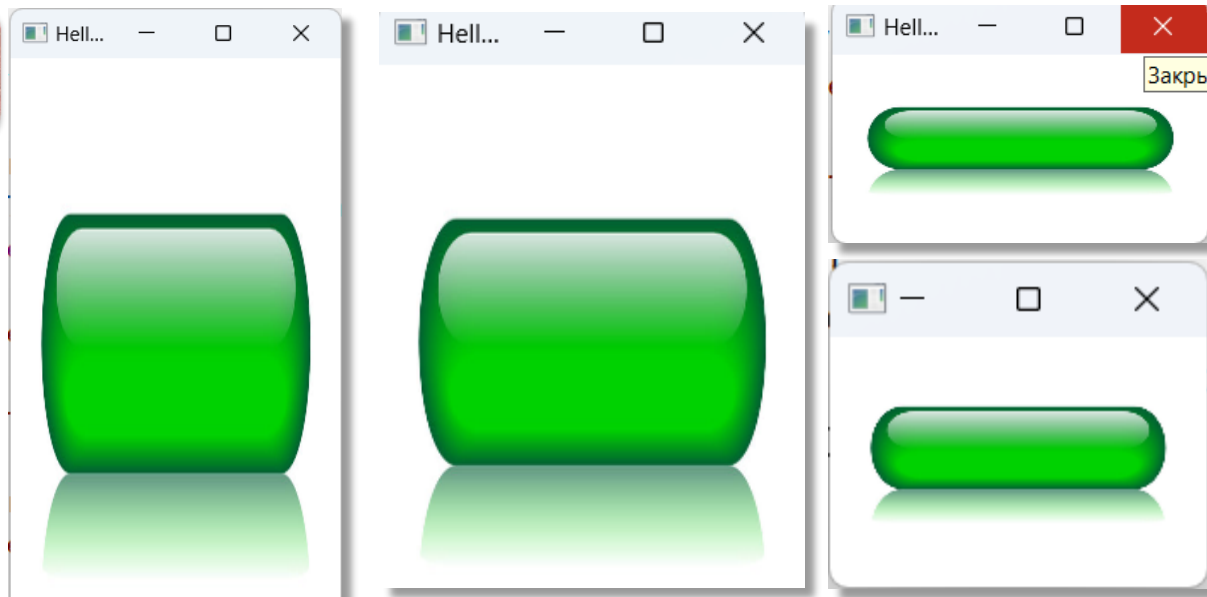

5. BorderImage

Элемент **BorderImage** позволяет разбить изображение на девять частей. Это бывает необходимо для создания *масштабируемой графики*. Основные трудности с изменяемыми размерами возникают у элементов, имеющих закругленные углы. Благодаря элементу **BorderImage** можно создать базовые компоненты, которые будут принимать различные размеры без искажений.



```
import QtQuick 2.15
BorderImage {
    source: "qrc: /button. png"
    width: 100
    height: 45
    border {
        left: 30; top: 15; right: 30; bottom: 15
    }
}
```

Изображение
масштабируемой
кнопки

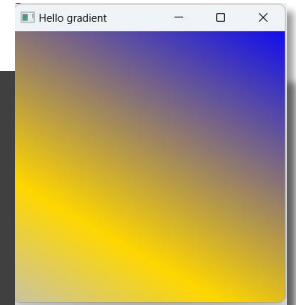


6. Градиенты

В язык *QML* включен только один градиент - линейный. Для его задания существует свойство **gradient**, которому в качестве значения необходимо присвоить элемент **Gradient**. Этот элемент содержит точки останова (элемент **GradientStop**). Каждая точка останова имеет позицию: номер между 0 (стартовая точка) и 1 (конечная точка) и цвет. *Стартовая и конечная* точки располагаются в верхних и нижних углах и не могут быть перемещены. Поэтому если нужно сделать линейный градиент по диагонали, то следует использовать элементы трансформации.

*Задание градиентов требует много ресурсов процессора. Поэтому желательно применять уже готовые изображения градиентов, вместо того, чтобы каждый раз создавать их. Градиенты можно создавать еще и при помощи элементов холста, а также с помощью модуля **QtGraphicalEffects**, который дает возможность создания линейных градиентов, конических, радиальных.

```
import QtQuick 2.15
Rectangle {
    width: 200
    height: 200
    gradient: Gradient {
        GradientStop {position: 0.0; color: "blue"}
        GradientStop {position: 0.7; color: "gold"}
        GradientStop {position: 1.0; color: "silver"}
    }
    rotation: 30;
    scale: 1.5
}
```

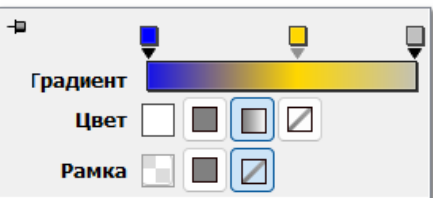


Интерактивная
настройки
градиентов.
позиция свойства
gradient, выберите
«Показать панель Qt
Quick».

height: 300

gradient:

Gradient



}

GradientStop {

position: 0.56;

color: "#ffd700";

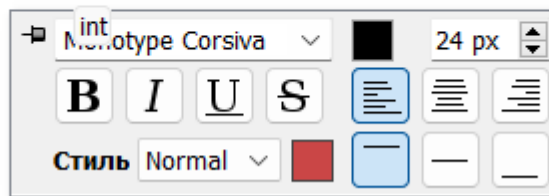
}

7. Шрифты

Все свойства настройки шрифтов расположены в группе **font**.

```
Text {  
...  
    font {  
        family: "Helvetica"  
        pixelSize: 24  
        bold: true  
    }  
...  
}
```

```
font {  
    family: "Monotype Corsiva"  
    pixelSize: 24  
}
```

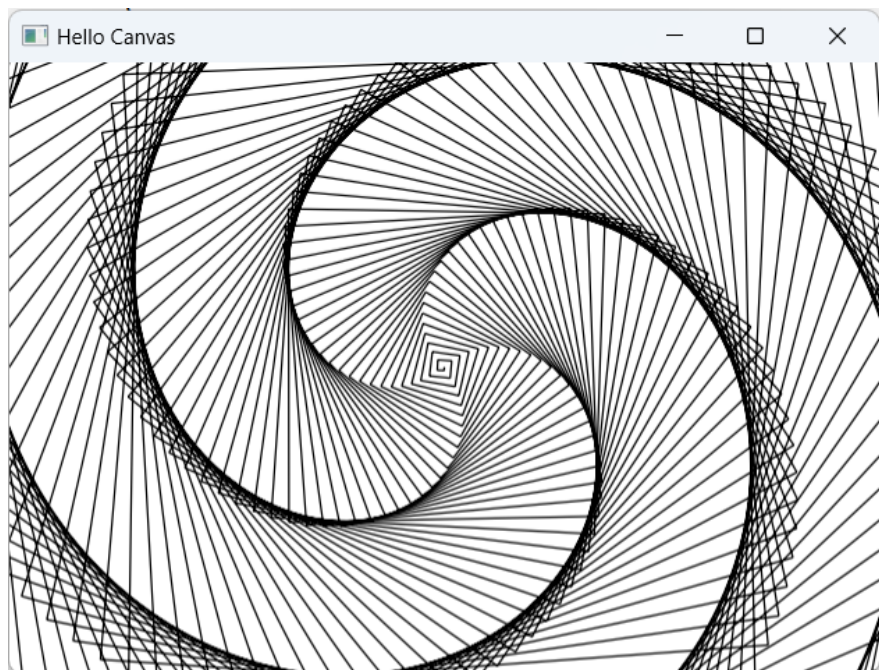


No clicks

*Изменение свойств шрифтов можно использовать интерактивное окно программы *Qt Creator*. Для его вызова просто поместите курсор мыши на область свойства группы **font**, нажмите правую кнопку мыши и в открывшемся контекстном меню выберите пункт «Показать панель Qt Quick»

8. Рисование на холсте

QML – позволяет реализовать алгоритмы для рисования, и для этой цели используется встроенный язык **JavaScript**. Элемент **Canvas** предоставляет свойство обработки **onPaint**, которое можно сравнить с событием *QWidget::paintEvent()*. Внутри этого свойства необходимо реализовать алгоритм рисования на *JS*.



Объект
контекста
рисования

```
import QtQuick 2.15
Window {
    width: 500; height: 350
    visible: true; title: qsTr("Hello Canvas")
    Canvas {
        anchors.fill: parent
        onPaint: {
            function drawFantasy() {
                ctx.beginPath()
                ctx.translate(parent.width / 2, parent.height / 2)
                var fAngle = 91 * 3.14156 / 180
                for (var i = 0; i < 300; ++i) {
                    var n = i * 2
                    ctx.moveTo(0, 0)
                    ctx.lineTo(n, 0)
                    ctx.translate(n, 0)
                    ctx.rotate(fAngle)
                }
                ctx.closePath()

                var ctx = getContext("2d");
                ctx.clearRect(0, 0, parent.width, parent.height)
                ctx.save();
                ctx.strokeStyle
                ctx.lineWidth = 1
                drawFantasy();
                ctx.stroke();
                ctx.restore();
            }
        }
    }
}
```

Элемент **Canvas** представляет собой элемент холста, на котором можно выполнять растровые операции.

Заполняется все
пространство окна

JavaScript
в работе

9. Вывод текста на холст

```
import QtQuick 2.15
Canvas {
    id: canv
    width: 400
    height: 160
    onPaint: {
        var ctx = getContext ( "2d" )
        ctx.fillStyle = "Black"
        ctx.fillRect(0, 0, canv.width, canv.height);
        ctx.strokeStyle = "Yellow"
        ctx.shadowColor = "Yellow";
        ctx.shadowOffsetY = 5;
        ctx.shadowBlur = 5;
        ctx.font = "48px Arial";
        ctx.fillStyle = "Yellow";
        ctx.fillText("Text has a shadow !", 10, canv.height / 2);
    }
}
```

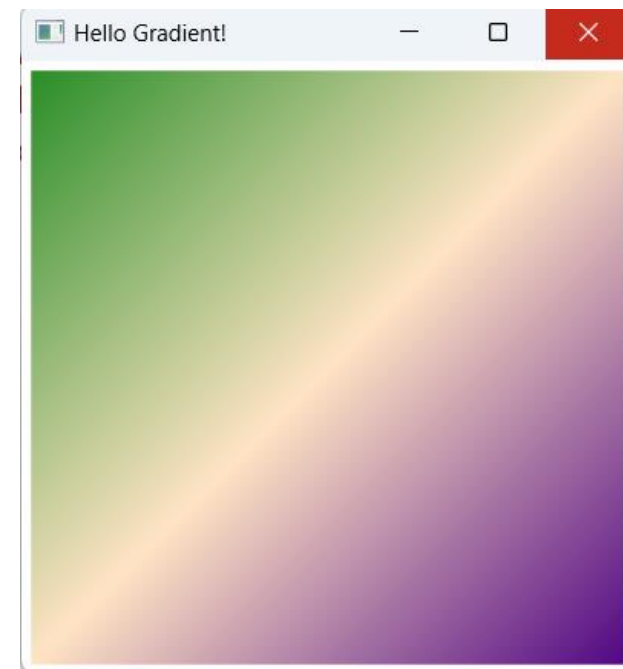
Для вывода текста на холст можно использовать метод ***fillText()***. Можно так же добавить несколько стилевых эффектов.



10. Линейные градиенты на холсте

```
import QtQuick 2.15
Canvas {
    id: canv
    width: 320
    height: 320
    onPaint: {
        var ctx = getContext ( "2d" )
        ctx.strokeStyle = "White"
        ctx.lineWidth = 15
        var gradient =
            ctx.createLinearGradient(canv.width, canv.height, 0, 0)
        gradient.addColorStop(0, "Indigo")
        gradient.addColorStop(0.5, "Bisque")
        gradient.addColorStop ( 1, "forestGreen")
        ctx.fillStyle = gradient
        ctx.fillRect(0, 0, canv.width, canv.height)
        ctx.strokeRect(0, 0 , canv.width, canv.height)
    }
}
```

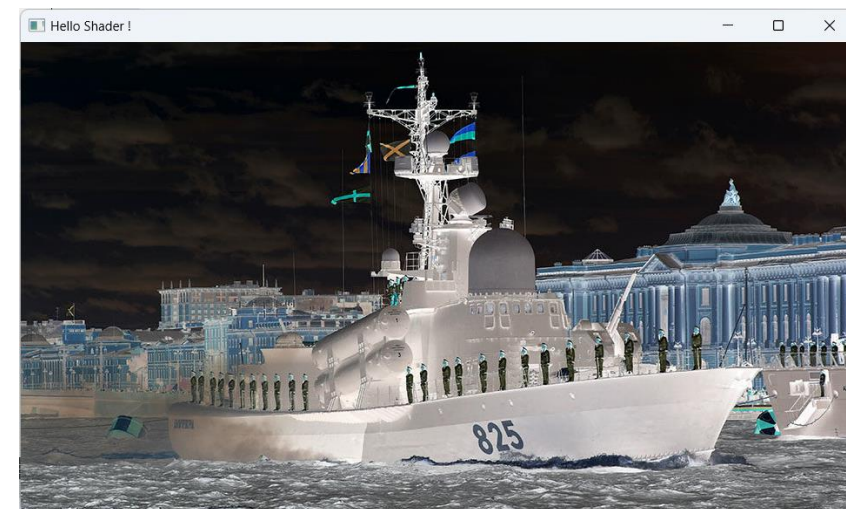
При помощи элемента холста можно отображать линейные градиенты.



11. Shader effects

Шейдер - программа, предназначенная для исполнения не (CPU), а процессором видеокарты (GPU). Сама программа написана на языке GLSL (*OpenGL Shading Language*). Это диалект **языка C**, обладающий целым рядом особенностей. Мощь шейдеров в скорости выполнения графических алгоритмов. Шейдеры работают с текстурами и фрагментами растровых изображений.

Для использования шейдеров *Qt Quick* предоставляет элемент **ShaderEffect**. Этот элемент переводит источник растрового изображения в текстуру и обрабатывает её при помощи заданного алгоритма шейдера, а затем отображает получившуюся новую текстуру. На рисунке показано применение шейдера с алгоритмом инвертирования.



12. «Procter and Gamble»

```
import QtQuick 2.15
```

```
Rectangle {
```

```
  id: rect
```

```
  width: sourceImage.width
```

```
  height: sourceImage.height
```

```
  color: "Black "
```

```
  ShaderEffect {
```

```
    ...
```

```
    width: sourceImage.width
```

```
    height: sourceImage.height
```

```
    property variant source: sourceImage
```

```
    ...
```

```
  }
```

```
}
```

```
Image {
```

```
  id: sourceImage
```

```
  visible: false
```

```
  source: "qrc:/picture.png"
```

```
}
```

Фактически мы смешиваем
два исходных кода вместе:
Код *QML* с кодом шейдера.

```
fragmentShader: "
```

```
  uniform sampler2D source;
```

```
  uniform lowp float qt_Opacity;
```

```
  varying highp vec2 qt_TexCoord0;
```

```
  void main() {
```

```
    gl_FragColor =
```

```
      abs(texture2D(source, qt_TexCoord0) *
```

```
        qt_Opacity - 1.0);
```

```
  }"
```

13. Библиотека эффектов

На базе элемента шейдера в *Qt Quick* реализована целая библиотека эффектов ***QGraphicalEffects***. Можно посмотреть их исходный код, использовать его в качестве учебного материала или для экспериментов, применять эти эффекты в своих программах. Список доступных эффектов в версии 1.0 модуля ***QGraphicalEffects*** приведён ниже.

Blend – Смешивает два источника изображений вместе.

BrightnessContrast – регулировка яркости и контраста.

ColorOverlay – изменяет цвет элемента, применяя к нему цвет наложения

Colorize – Устанавливает цвет в пространстве HSL – модели

ConicalGradient – рисует конический градиент

Desaturate – уменьшает насыщенность цветов

DirectionalBlur – Эффект размытия в указанном направлении

Displace – перемещает пиксеты исходного элемента в соответствии с указанным источником смещения

DropShadow – рисует тень за исходным элементом

FastBlur – быстрый эффект размытия

GammaAdjust – регулировка яркости

GaussianBlur – эффект размытия высокого качества

Glow – генерирует эффект сияния вокруг исходного элемента

14. Библиотека эффектов

HueSaturation –Изменяет цвета источника в пространстве HSL- модели

InnerShadow – рисует внутреннюю тень

LevelAdjust – регулировка уровней цвета в пространстве HSL - модели

LinearGradient – линейный градиент

MaskedBlur – Эффект размытия с изменяющейся интенсивностью

OpacityMask – маскирует исходный элемент другим элементом

RadialBlur – направленное размытие в круговом направлении вокруг центральной точки

RadialGradient – радиальный градиент

RectangularGlow – эффект свечения прямоугольной области

ResursiveBlur – сильное размытие

ThresholdMask – маскирует исходный элемент другим элементом и применяет пороговое значение

ZoomBlur – направленный эффект размытия, который применяется к центральной точке источника.

```
import QtQuick 2.15
import QtQuick.Controls 2.2
import QtGraphicalEffects 1.0
Column {
    FastBlur {
        id: blur
        Image {
            id: sourceImage
            visible: false
            source: "qrc: /apple.jpg"
        }
        width: sourceImage.width;
        height: sourceImage.height
        source: sourceImage
    }
    Slider {
        id: sld
        width: sourceImage.width
        value: 0; from: 0; to: 64; stepSize: 1
        onValueChanged: {
            blur.radius = value
        }
    }
}
```

15. Пример – «Эффект размытия»



16. Домашка # 9

Организовать приложение для изучения библиотеки эффектов ***QGraphicalEffects***.
В приложении должна быть возможность смешивать эффекты, используемые в библиотеки(за основу можно взять приложение со слайда 15.)