# Векторная графика

Конечной целью графического приложения является отображение интерфейса пользователя на экране и обеспечение взаимодействия с пользователем.

Типовое устройство вывода — экран — является растровым. Однако отображаемые в окне элементы интерфейса и шрифты зачастую описываются векторно.

Процесс перевода векторных объектов в растровую картинку, отображаемую на экране, носит название **рендеринг** или **визуализация**.

# Графика Qt

В системе Qt для осуществления рендеринга существует специальный класс **QPainter**. Он содержит методы рисования различных геометрических примитивов и текста.

Вспомогательными классами для QPainter являются:

- QPoint, QPointF — координаты точки в пространстве;
- QSize, QSizeF — размеры прямоугольного объекта, ширина и высота;
- QRect, QRectF — координаты прямоугольной области;
- QColor — цвет отображаемого объекта
- QPen — стиль рисования обводки;
- QBrush — стиль рисования заливки;
- QFont — описание векторного шрифта.

# Экосистема

Графическое приложение может вызывать функции отрисовки только при обработке **события рисования**. Данное событие порождается операционной системой.

```
#include <QWidget>
#include <QPainter>
class Paintee : public QWidget
{
protected:
    void paintEvent(QPaintEvent* e)
    {
        QPainter p(this);
    }
}
```

# Экосистема

При необходимости можно запросить событие рисования для виджета, вызвав слот **repaint** или **update**.

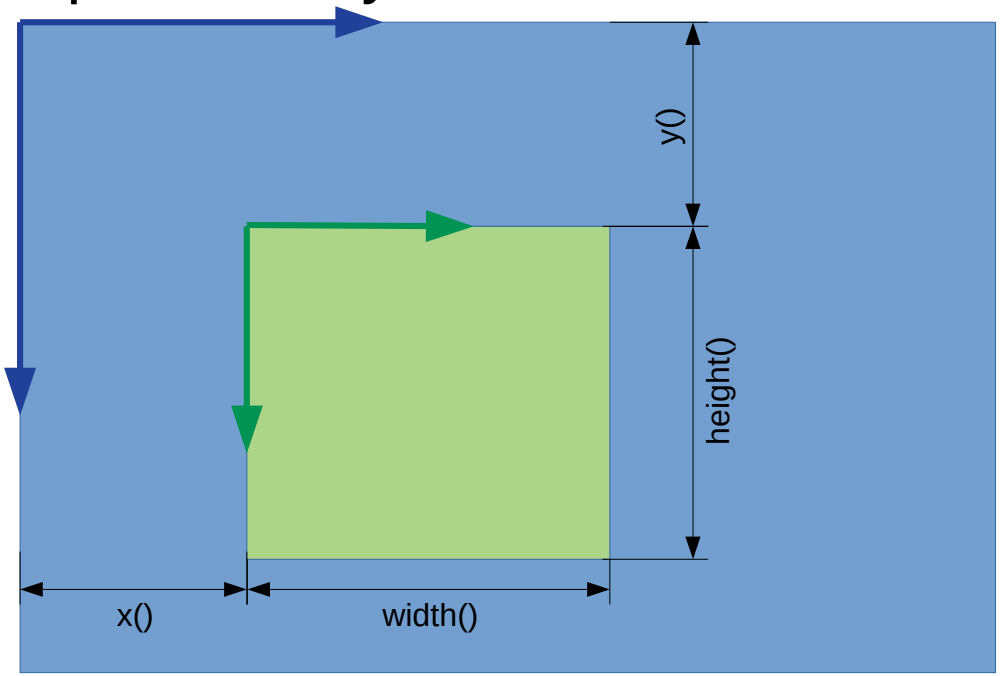
```
class Paintee : public QWidget
{
private slots:
    void onTimer()
    {
        repaint(); // Запрос события рисования
    }
protected:
    void paintEvent(QPaintEvent* e)
    {
        QPainter p(this);
    }
}
```

# Система координат

При рисовании используются экранные координаты: ось X смотрит вправо, ось Y — вниз.

Точка с координатами (0, 0) расположена в левом верхнем углу виджета.

Координаты самого виджета задаются относительно родительского, а при его отсутствии — относительно экрана.



# Координаты

Класс **QPoint** хранит точку в экранных координатах в виде пары целых чисел. **QPointF** хранит координаты в вещественных числах. Классы поддерживают арифметические операции.

```
QPoint p1(3, 12);
QPoint p2 = p1 + QPoint(1, -1);
cout << p2.x() << ":" << p2.y();                    // 4:11
```

**QSize** и **QSizeF** хранят размеры в пикселях: ширину и высоту.

```
QSizeF sz(640, 480);
cout << sz.width() << "x" << sz.height();   // 640:480
```

**QRect** и **QRectF** совмещают в себе координаты и размеры, определяя прямоугольную область. Класс содержит методы для вычисления координат любой характерной точки прямоугольника.

```
QRect r = ui->pushButton->geometry();
r.setWidth(180);
cout << r.bottomRight().x() << r.bottomRight().y();
```

# Цвет

Цвет описывается классом **QColor**.

Он может работать в пространствах цветов RGB, HSV и CMYK и может быть преобразован между ними.

```
QColor c(255, 0, 0);                    // Красный цвет
c = QColor::fromHsv(120, 128, 255);     // Бледно-зеленый
```

Также доступны предустановленные цвета Qt

```
QColor c(Qt::yellow);                   // желтый
```

и возможность задать цвет в формате, используемом в Web-страницах.

```
QColor g("#00FF00");                    // Зеленый
QColor b("indigo");
cout << b.blue() << endl;               // 130
```

Цвета Qt поддерживают альфа-канал: 255 — полностью непрозрачный, 0 — полностью прозрачный.

# Перо и кисть

Перо **QPen** определяет способ рисования линий. На уровне пера определяются цвет, стиль и толщина линии. По умолчанию выбирается сплошная линия шириной в 1 пиксель.

```
QColor clr(0, 255, 0);
QPen pen(clr);
pen.setLineWidth(3);          // Зеленая линия, 3 пикселя
```

Кисть **QBrush** определяет способ заполнения замкнутых областей. На уровне кисти определяется цвет и стиль заливки. По умолчанию заливка идет сплошным цветом. При необходимости можно обеспечить заливку текстурой или градиентом.

```
QColor bg(128, 128, 128);
QBrush brush(bg);
brush.setStyle(Qt::FDiagPattern); // Диагональные линии
```

Константы Qt::NoPen и Qt::NoBrush определяют отсутствие обводки и заливки соответственно.

# Градиент

Градиент QGradient представляет собой способ расчета заливки для кисти. Производные классы **QLinearGradient** и **QRadialGradient** определяют линейный и круговой градиент соответственно.

Внутри градиента цвет линейно интерполируется от значения в точке 0 до значения в точке 1. Метод setColorAt позволяет задать цвет в пределах единичного интервала, относительно которого будет производиться интерполяция.

```
QLinearGradient bg_gradient(0, 0, width(), height());
bg_gradient.setColorAt(0, QColor(0, 0, 0));
bg_gradient.setColorAt(1, QColor(128, 128, 128));
QBrush gradient_brush(bg_gradient);
```

# Шрифт

Класс **QFont** определяет шрифт. На уровне этого класса задаются имя шрифта (например, Times New Roman), кегль, жирность, наклон и т.д. QPainter и виджеты содержат в себе шрифт.

```
QPainter pnt(this);
QFont fnt = pnt.font();
fnt.setPixelSize(24);    // 24 пикселя в высоты
pnt.setFont(fnt);
```

Вспомогательный класс **QFontMetrics** позволяет получить размеры конкретной строки в переданном шрифте в пикселях.

```
QFontMetrics fm(fnt);
cout << fm.height() << ", " << fm.width("Привет!");
// 13, 37
```

# Рисование

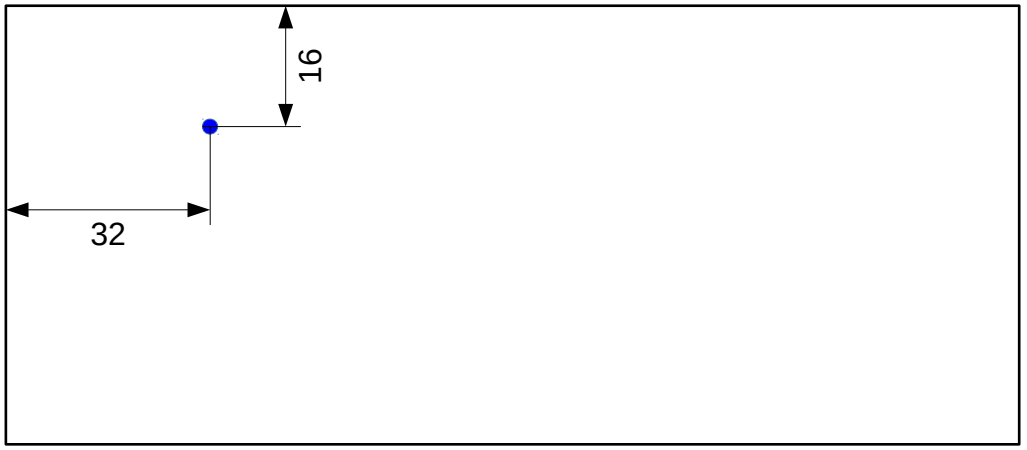
Рисование при помощи QPainter происходит по следующему алгоритму:

- Создается экземпляр класса QPainter. Аргументом конструктора при создании является объект, на котором происходит рисование: виджет или изображение.
- Определяются необходимые цвета QColor.
- Создается перо QPen и кисть QBrush. Перо и кисть передаются в QPainter через методы setPen и setBrush. Перо и кисть будут использоваться во всех процедурах рисования до задания нового пера или кисти.
- Вызывается один из методов draw* из класса QPainter. Аргументы метода — координаты точек, через которые будет проходить нарисованный объект.

# Методы рисования

- drawPoint: рисует точку с переданными координатами тем способом, который определен пером. Кисть игнорируется.

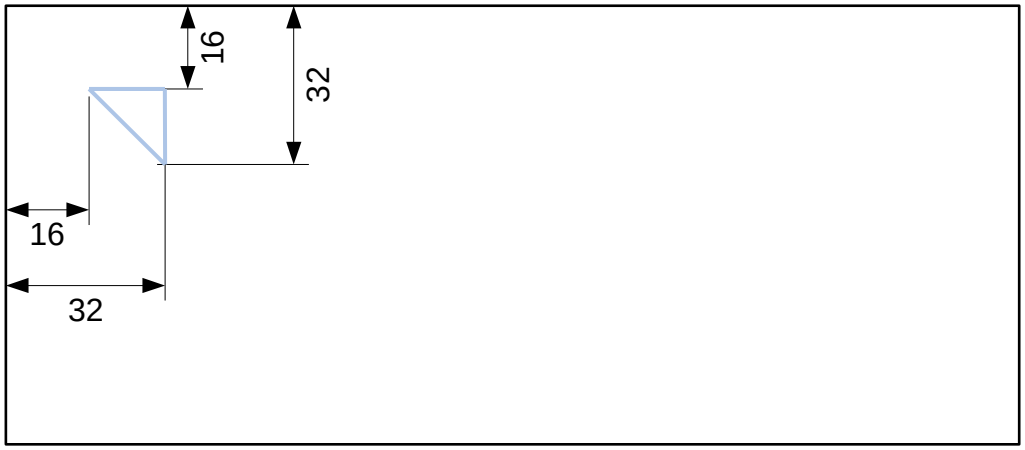
```
QPainter pnt(this);
QPen p(QColor(0, 0, 128));
p.setWidth(3);
pnt.setPen(p);
pnt.drawPoint(32, 16);
```



# Методы рисования

- drawLine: рисует прямую линию между переданными координатами тем способом, который определен пером. Кисть игнорируется.

```
pnt.drawLine(16, 16, 32, 16);
pnt.drawLine(32, 16, 32, 32);
pnt.drawLine(32, 32, 16, 16);
```



# Методы рисования

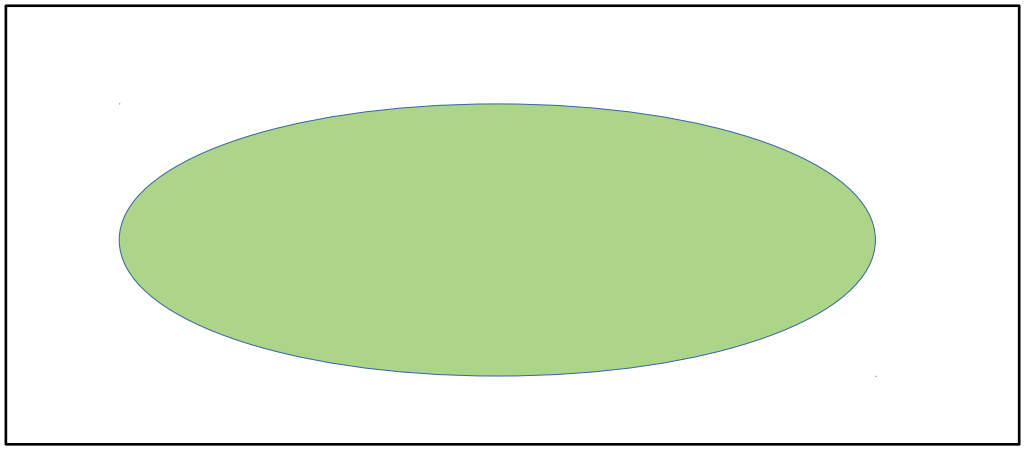
- drawRect: рисует прямоугольник с переданными координатами. Обводка и заливка прямоугольника регулируются пером и кистью.

```
pnt.setPen(QPen(QColor(0, 0, 0)));
pnt.setBrush(QBrush(QColor(173, 213, 138)));
pnt.drawRect(20, 20, 200, 100);
```

# Методы рисования

- drawEllipse: рисует эллипс. Координаты могут задаваться прямоугольником, в который вписан эллипс, либо координатами центра и размерами полуосей. Обводка и заливка эллипса регулируются пером и кистью.

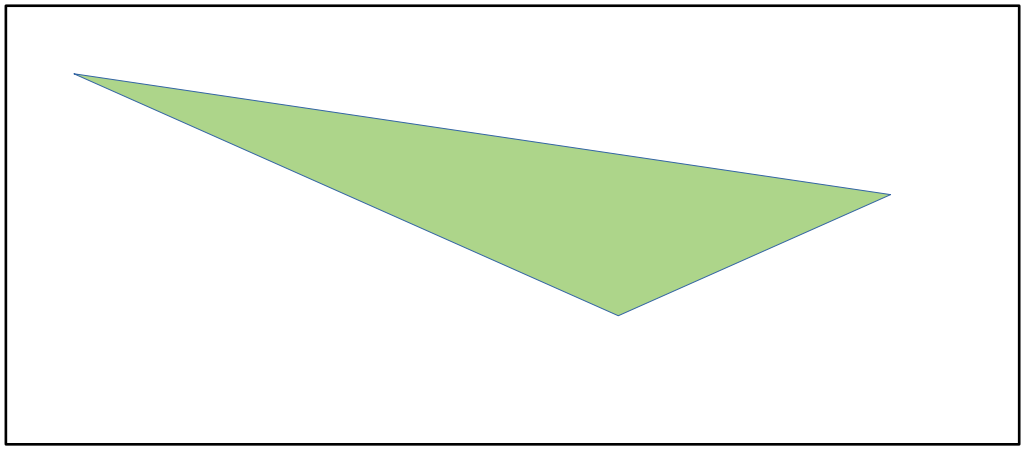
```
pnt.setPen(QPen(QColor(0, 0, 0)));
pnt.setBrush(QBrush(QColor(173, 213, 138)));
pnt.drawEllipse(20, 20, 200, 100);
```



# Методы рисования

- drawPolygon: рисует произвольный многоугольник, определенный как массив его вершин. Последняя вершина соединяется с первой. Обводка и заливка полигона регулируются пером и кистью.

```
QPoint pts[4] = { QPoint(10, 10), QPoint(200, 50),
                  QPoint(150, 100)};
pnt.drawPolygon(pts, 3);
```



# Методы рисования

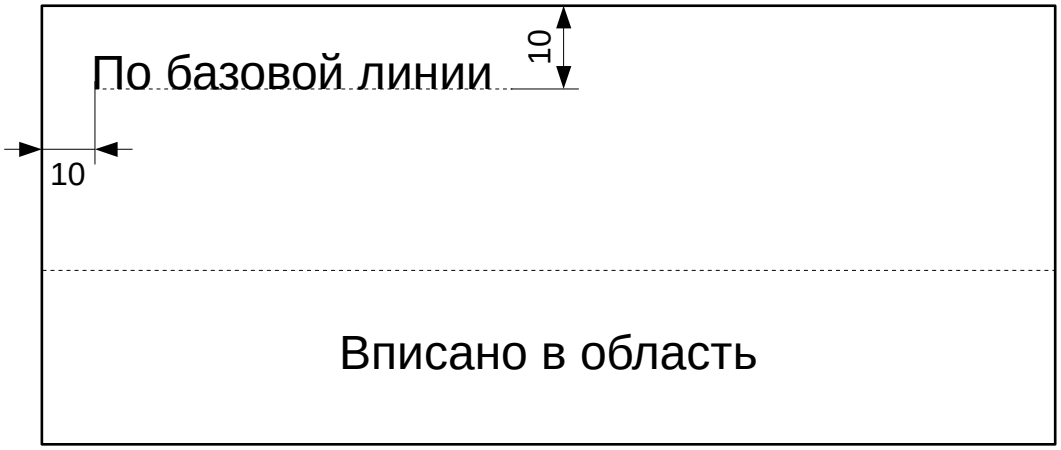
- drawImage: рисует изображение, хранящееся в QImage. Изображение либо располагается левым верхним углом в переданных координатах, либо вписывается в переданный прямоугольник с масштабированием. Кисть и перо игнорируются.

```
QImage img("рысь.jpg");
pnt.drawImage(QPoint(0, 0), img);
pnt.drawImage(QRect(100, 0, 50, 50), img);
```

# Методы рисования

- drawText: рисует текст. При рисовании либо указываются координаты базовой точки, либо прямоугольник и способ вписывания текста в область. Способ рисования текста определяется пером, кисть игнорируется.

```
pnt.drawText(10, 10, "По базовой линии")
pnt.drawText(QRect(0, 100, width(), 50),
             Qt::AlignCenter, "Вписано в область");
```



# Обрезка

QPainter позволяет ограничить область рисования при помощи метода setClipRect(QRect). Для отключения обрезки необходимо вызвать setClipping(false);

```
pnt.setClipRect(QRect(0, 0, width() / 2, height());
pnt.drawEllipse(20, 20, 200, 50);
pnt.setClipping(false);
pnt.drawRect(20, 50, 200, 100);
```