2D графика. QPainter

- Класс QPainter выполняет низкоуровневое рисование на виджетах и других устройствах рисования.
- QPainter может рисовать простые геометрические фигуры: точки, линии, прямоугольники, эллипсы, дуги, сегменты круга, замкнутые ломаные (многоугольники) и кривые Безье. Он так же может отображать карты пикселей, рисунки и текст.

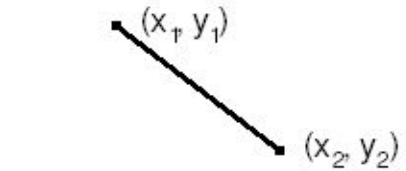
- Тремя наиболее важными характеристиками QPainter являются перо (pen), кисть (brush) и шрифт (font).

- Перо используется для рисования линий и границ геометрических фигур. Оно характеризуется такими параметрами, как: цвет, толщина, стиль рисования линий, стиль оформления концов линий и стиль оформления углов. setPen()

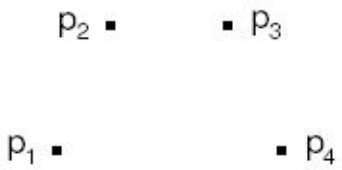
- Кисть - это шаблон, которым заполняются геометрические фигуры. Кисти характеризуются цветом и стилем. setBrush()

- Шрифт используется для рисования текста. Шрифт может иметь огромное количество атрибутов, среди них: название и размер. setFont()

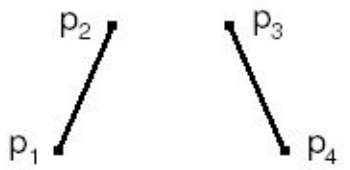
Методы класса QPainter, для рисования геометрических фигур



drawLine()

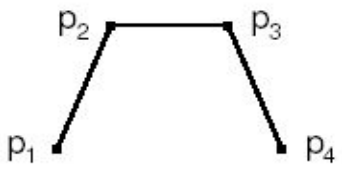


drawPoints()

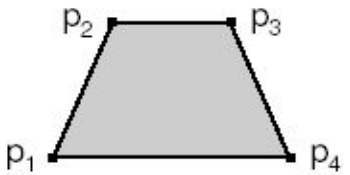


drawLineSegments()

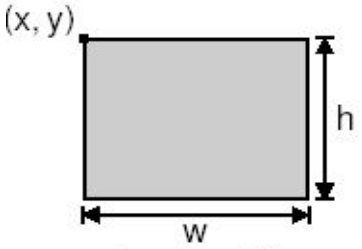
drawCubicBezier()



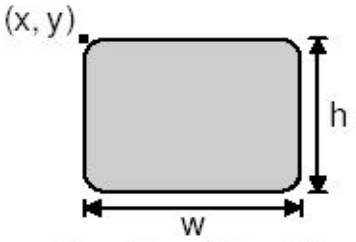
drawPolyline()



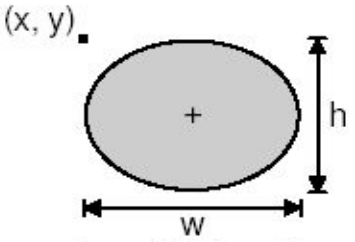
drawPolygon()



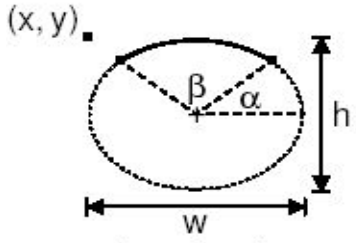
drawRect()



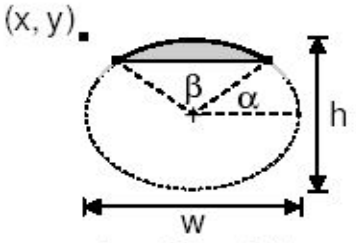
drawRoundRect()



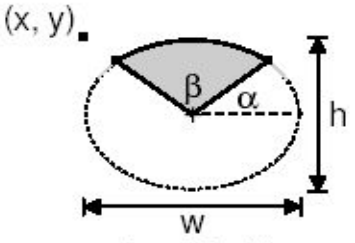
drawEllipse()



drawArc()



drawChord()

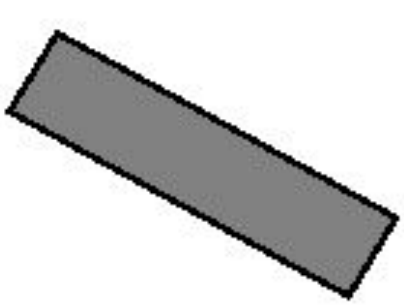


drawPie()

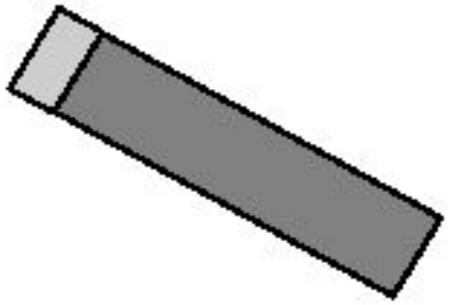
Стили пера.


Толщина линий
1
2
3
4
NoPen
SolidLine
DashLine
DotLine
DashDotLine
DashDotDotLine

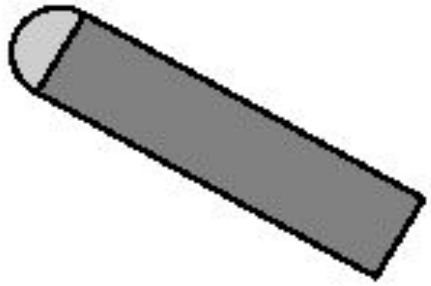
Стили оформления концов линий и углов.



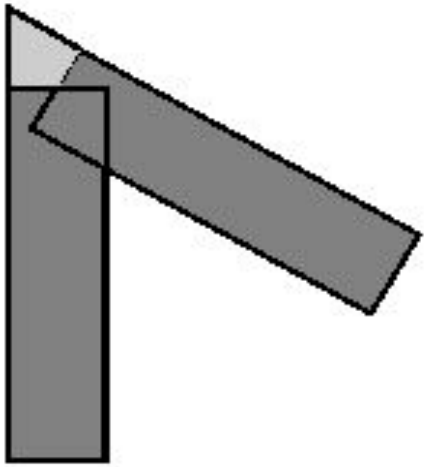
FlatCap



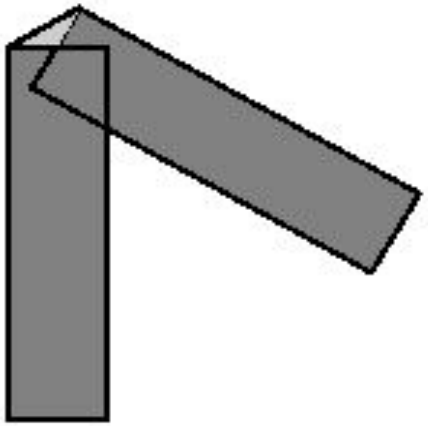
SquareCap



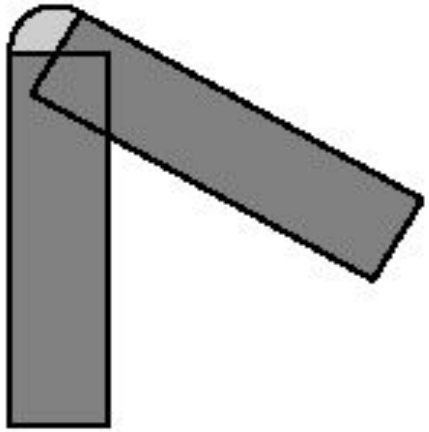
RoundCap



MiterJoin

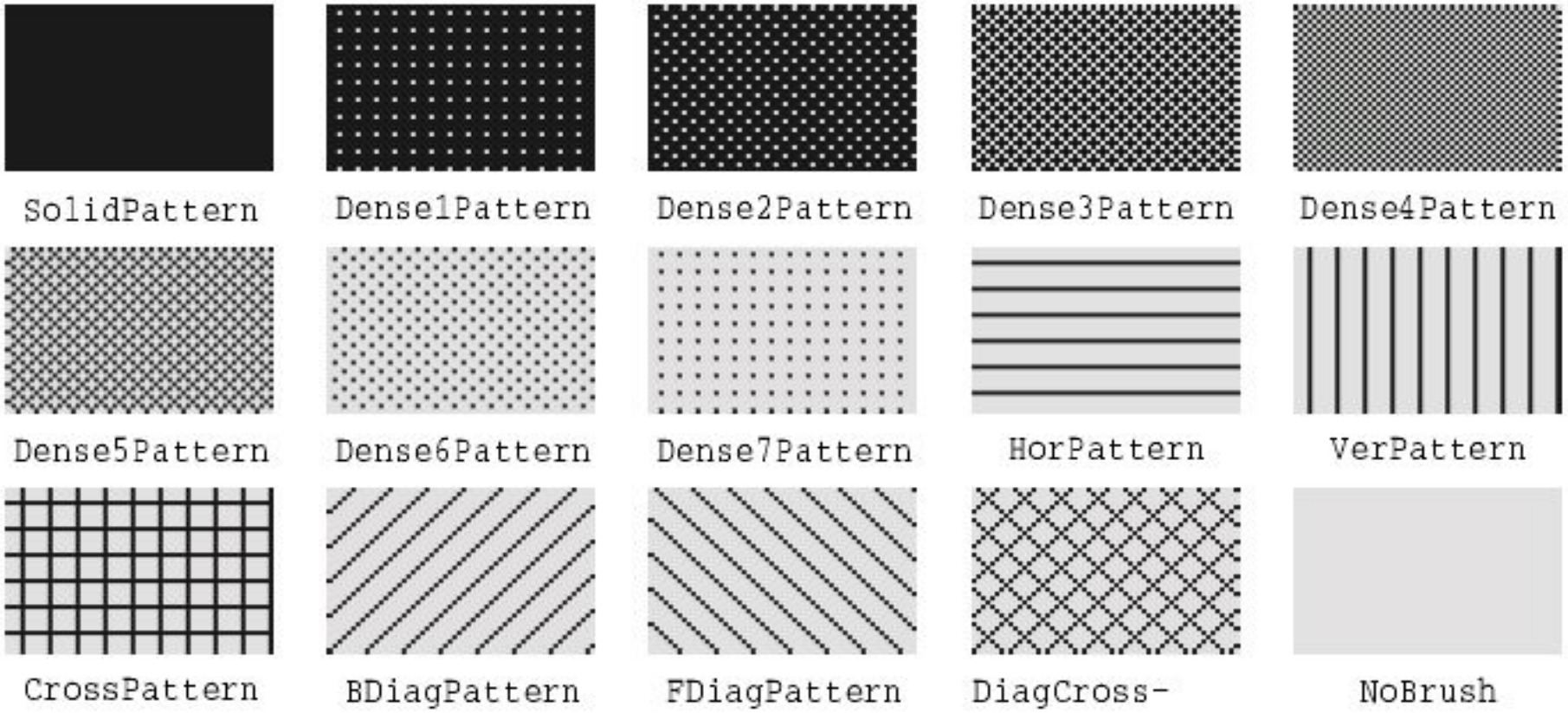


BevelJoin



RoundJoin

Стили кисти.


SolidPattern
Dense1Pattern
Dense2Pattern
Dense3Pattern
Dense4Pattern
Dense5Pattern
Dense6Pattern
Dense7Pattern
HorPattern
VerPattern
CrossPattern
BDiagPattern
FDiagPattern
DiagCross-
NoBrush

```cpp
#include <QtGui/QApplication>
#include "mainwindow.h"
#include <QPainter>
#include <QLabel>

class MyWindow : public QMainWindow {
protected:
        virtual void paintEvent(QPaintEvent* e) {
                QMainWindow::paintEvent(e);

                QPainter p(this);
                p.setPen(QPen(Qt::red, 2, Qt::DotLine));
                p.drawLine(0, 0, 100, 100);
        }
};

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MyWindow w;
    w.show();
    return a.exec();
}
```

```
QPainter p(this);
p.setPen(QPen(Qt::black, 3, Qt::DashDotLine));
p.setBrush(QBrush(Qt::green, Qt::SolidPattern));
p.drawEllipse(20, 20, 100, 60);
```

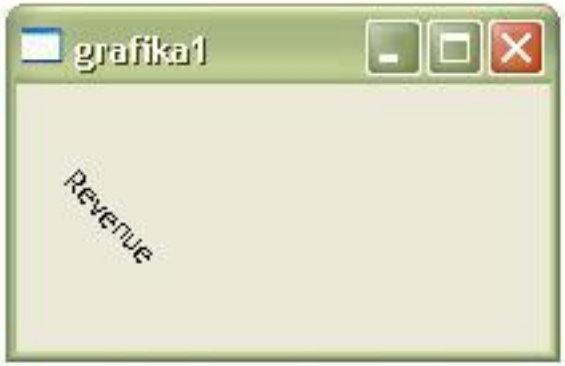
```
#include <QtGui/QApplication>
#include "mainwindow.h"
#include <QPainter>
#include <QPushButton>
#include <QMatrix>

class MyButton : public QPushButton {
protected:
        virtual void paintEvent(QPaintEvent* e) {
                QPushButton::paintEvent(e);
                QPainter p(this);
                p.setPen(QPen(Qt::black, 1, Qt::SolidLine));
                p.setBrush(QBrush(Qt::green, Qt::SolidPattern));
                p.drawEllipse(10, 7, 10, 6);
        }
};

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MyButton b;
    b.show();
    return a.exec();
}
```

Параметры системы координат

- **область просмотра** (viewport) - это произвольный прямоугольник, заданный физическими координатами.
- **окно** (window) - описывает тот же самый прямоугольник, но уже в логических координатах.
- **матрица преобразования** (world matrix) задает набор трансформаций, которые должны быть выполнены в дополнение к преобразованиям логических координат в физические.

- По-умолчанию координаты **области просмотра** и **окна** совпадают с системой координат физического устройства. Например, если устройство отображения представляет из себя виджет, с размерами 320 X 200, то и **область просмотра** и **окно** имеют те же самые размеры. В данном случае логическая и физическая системы координат совпадают.

- **матрица преобразования** позволяет выполнять изменение масштаба, вращение и сдвиг рисуемых элементов. Например, если необходимо нарисовать текст под углом 45 градусов, то можно написать следующий код:

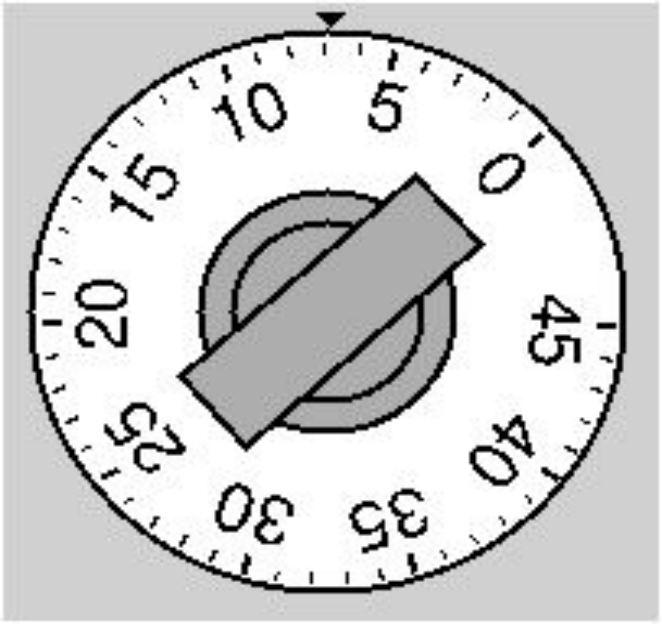



```
QMatrix matrix;
matrix.rotate(45.0);
matrix.translate(-40.0, -40.0);
p.setWorldMatrix(matrix);
p.drawText(rect(), Qt::AlignCenter, tr("Revenue"));
```

При необходимости, матрицу преобразований можно сохранить вызовом saveWorldMatrix() и затем восстановить вызовом restoreWorldMatrix().

Реализация Таймера электропечи:

- http://www.opennet.ru/docs/RUS/qt3_prog/c4100.html



2D графика. QCanvas

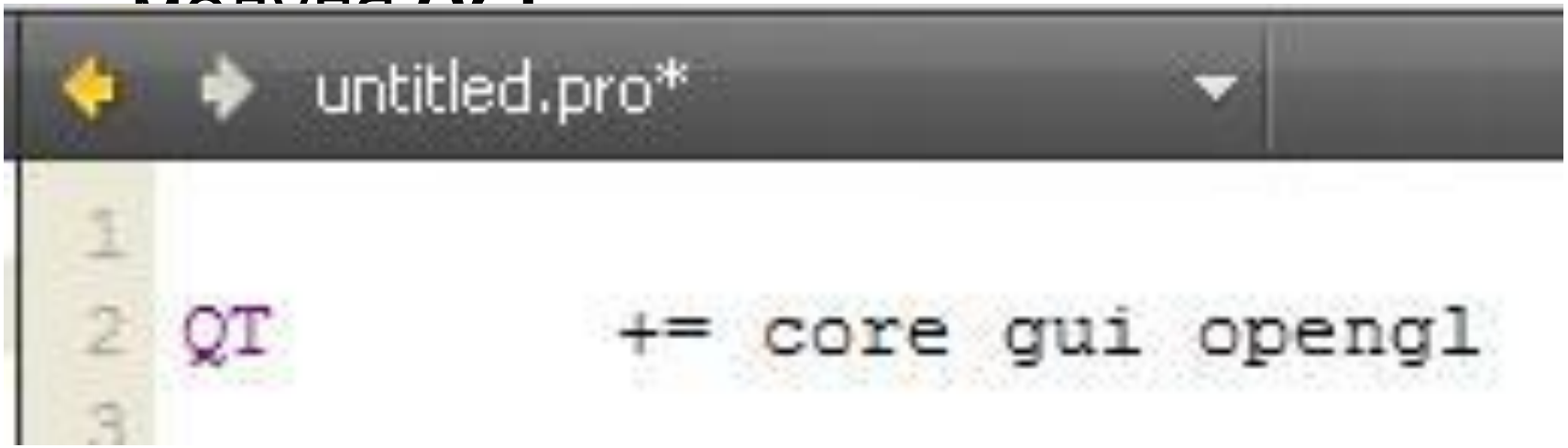
- **QCanvas** (Canvas -- холст, полотно, канва. прим. перев.) предоставляет более высокоуровневый интерфейс, чем **QPainter**. Он может включать в себя элементы любой формы и имеет внутреннюю реализацию двойной буферизации (когда изменяется какой либо элемент, то перерисовывается только та часть, которая действительно изменилась).

- Элементы, которые может отображать **QCanvas**, являются экземплярами класса **QCanvasItem** или его потомков.
- Qt содержит неплохой набор предопределенных графических элементов: *QCanvasLine, QCanvasRectangle, QCanvasPolygon, QCanvasPolygonalItem, QCanvasEllipse, QCanvasSpline, QCanvasSprite и QCanvasText..*

- Классы **QCanvas** и **QCanvasItem** - просто данные, они не имеют визуального представления.
- Для отображения **QCanvas** и его элементов мы должны использовать ***виджет QCanvasView***. Такое разделение данных и средств их отображения, позволяет отображать один и тот же **QCanvas** в нескольких **QCanvasView**, причем каждый из них может визуализировать свою собственную часть **QCanvas**, причем с применением различных матриц преобразования. http://www.opennet.ru/docs/RUS/qt3_prog/x4318.html

Графика OpenGL

- **OpenGL** - это стандарт API, для отображения двух- и трехмерной графики. Приложения Qt могут использовать OpenGL, посредством модуля **QGL**



mainwindow.h* | MainWindow(QWidget * =

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3   #include <QGLWidget>
4   #include <QtOpenGL>
5   #include <QTimer>
6
7   class MainWindow : public QGLWidget
8   {
9       Q_OBJECT
10
11  public:
12      MainWindow(QWidget *parent = 0);
```

QMainWindow — класс для вывода простого окна, а т.к. мы будем работать с opengl, нам понадобится **QGLWidget** — это класс для вывода графики, реализующий функции библиотеки **OpenGL**.

- **QGLWidget** при первой инициализации класса вызывает методы в следующем порядке:
- При запуске: **initializeGL()->resizeGL()->paintGL()**
- При изменении размера окна: **resizeGL()->paintGL()**
- updateGL() вызывает **paintGL()**

- **initializeGL** — необходимо использовать для глобальных настроек построения изображения, которые нет необходимости указывать при построении кадра.

- **resizeGL** — служит для построения размера окна. Если в ходе работы изменится размер окна, но не изменить область просмотра, то при увеличении размера можно наблюдать непредсказуемые явления.

- **paintGL** — этот метод будет выстраивать каждый кадр для отображения.

Двойная буферизация

- **PaintGL** не рисует сразу картинку на экран, а заносит в буфер, а по запросу **swapBuffers()** заменяет текущее изображение на то, что появилось в буфере.
- Буферция позволяет более корректно заменять изображение, чтоб не происходили скачки на экране.

События клика мыши

- **mousePressEvent()** — метод автоматически вызывается при нажатии клавиш мыши. В передаваемых параметрах можно получить различную информацию, например, какой именно кнопкой было сделано нажатие и по какой точке по координатам.

- *-Данное событие в нашем примере используется для определения куда кликнули мышью, затем если наши координаты находятся в поле квадрата, то добавляем к нашим очкам + 1 и перестраиваем наш кадр.*
- *-Так же используем для определения начальных координат для выделения области на экране, при зажатии и перемещении указателя.*

Событие перемещения указателя мыши

- **mouseMoveEvent()** — автоматически вызывается при изменении координат указателя мыши. По умолчанию установлено **setMouseTracking(false)**, поэтому событие вызывается только при условии нажатия клавиш мыши, для того, чтоб метод вызывался даже без нажатия необходимо установить **setMouseTracking(true).**
- *— Данный метод мы используем для получения текущего положения указателя*

Событие «отжатия» клавиши мыши

- **mouseReleaseEvent()** — автоматически вызывается при условии «отжатия» кнопки мыши. Так же принимает различные параметры.
- *— В данном случае мы используем метод, чтоб стереть с экрана выделенную нами область.*

Событие нажатия клавиш на клавиатуре

- **keyPressEvent()** — метод вызывается при событии, когда нажимается кнопка на клавиатуре.
- *— В нашем примере, мы используем этот метод, для того, чтоб переопределить координаты нашего квадрата и переместить его в новое место.*

Таймер

- **QTimer** — позволяет создать поток, который будет слушать сигналы и запускать соответственные слоты.
- *— В данном случае мы создаем таймер, который будет ждать 750мс, после чего он останавливается, отправляя сигнал* ***timeout()*** *, но мы при окончании сигнала будем не останавливать работу, а снова запустим слот на переопределение координат квадрата, по которому нужно кликать для того, чтоб набрать очки.*

Приложение «Куб» трехмерное

- http://www.opennet.ru/docs/RUS/qt3_prog/x4697.html