10. Лабораторная работа № 4 «Использование веб-камеры и пальцевого интерфейса сенсорного экрана»

10.1. Цель лабораторной работы

Демонстрация процесса разработки практического приложения, позволяющего рисовать пальцем на снимке с вебкамеры.

10.2. Инструкция по выполнению лабораторной работы

Основное устройство ввода на планшетном компьютере – сенсорный экран или кратко тачскрин. Первоначально тачскрин был вынужденной заменой широко распространенной «мыши», пользовательские интерфейсы для него требовали высокой точности клика, для чего требовался стилус. Затем произошло усовершенствование интерфейсов, их огрубление под пальцевый ввод, необходимость в стилусе отпала, но частичная взаимозаменяемость тачскрина и «мыши» сохраняется.

В этой лабораторной работы мы будем использовать тачскрин как холст художника – наносить пальцами штрихи на изображение. Для этого необходимы базовые знания языка C++, знакомство с основами фреймворка *Qt* и библиотекой MeeGoTouch. В качестве одной из составляющих приложения будут использованы материалы лабораторной работы «Захват видеоизображения» из первой части курса. Также необходим планшет, работающий под управлением Linux. Мы использовали MeeGo 1.1.99 Tablet на планшете 3Q TU1102T. На эмуляторе или виртуальной машине поддержки тачскрина скорее всего не будет, функции взаимодействия с

пользователем возьмет на себя «мышь» операционной системы.

Задача состоит из четырех частей – получение кадра от вебкамеры, отображение полученного кадра в пользовательском интерфейсе, использование информации от тачскрина для рисования и сохранение полученного рисунка в файл. Работа с вебкамерой подробно освещена в первой части курса, мы вернемся к ней в конце.

Рассмотрим задачу отображения графической информации в GUI. Библиотека MeeGoTouch содержит специальный виджет – `MImageWidget` – для этой цели. Достаточно добавить этот виджет в приложение и назначить ему изображение методом `setImage()`. Данный метод может принимать в качестве параметра `QImage` или `QPixmap`. Оба эти класса *Qt* предназначены для хранения изображений, но первый ориенитрован на чтение/запись изображений в различных форматах и попиксельный доступ к изображению для редактирования, а второй оптимизирован для вывода изображения на экран. Получить от `MImageWidget` можно только `QPixmap`, но не `QImage`. Естественно, существует возможность конвертации изображения из одного хранилища в другое. Кроме того, в *Qt* существует класс QBitmap, позволяющий хранить особый вид изображений – битовые маски, а также класс `QPicture`, который хранит изображение в форме последовательности команд для отрисовки.

Следующим шагом настройки `MImageWidget` будет установка его размеров в соответствии с размерами картинки.

```
widget.setMaximumSize(image.size());
widget.setMinimumSize(image.size());
widget.setPreferredSize(image.size());
```

Если этого не сделать, то компоновщик в некоторых ситуациях (например, изменение ориентации устройства) может изменить размеры виджета и координаты внутри виджета перестанут совпадать с координатами изображения. Возможно преобразовывать координаты на лету, но мы

вместо этого просто фиксируем размер виджета. В приложении создан класс `MyWidget` – наследник `MImageWidget`, к которому относится все вышенаписанное.

Теперь надо получить информацию от тачскрина. Большая часть информации внутри *Qt* передается в виде событий – наследников класса `QEvent`, попадающих в соответствующую очередь, а через нее – к виджетам и другим элементам приложения. События тачскрина относятся к классу `QTouchEvent` и имеют всего 3 типа – `QEvent::TouchBegin` (начало прикосновения), `QEvent::TouchUpdate` (движение по тачскрину), `QEvent::TouchEnd` (окончание прикосновения. По умолчанию они не доходят до нашего виджета в неизменном виде, система преобразует их к событиям более высокого уровня (например, `QGestureEvent`) и обрабатывает в другом месте. Для получения события тачскрина надо вызвать метод setAcceptTouchEvents(true).

Чтобы обрабатывать события, необходимо переопределить обработчик `bool MyWidget::event(QEvent *event)`. Такой обработчик есть у каждого виджета и вообще может быть реализован у любого наследника `QObject`. В новом обработчике мы проверяем тип события, и если оно относится к тачскрину – выполняем приведение типов, чтобы получить доступ к специфичной информации:

```
QTouchEvent *touchevent =
static_cast<QTouchEvent*>(event);
```

Для приведения типов мы используем конструкцию `static_cast`, использование dynamic_cast не рекомендуется, как и приведения в стиле чистого C. После обработки события следует сообщить очереди событий, что оно уже обработано и в дальнейшей обработке не нуждается:

```
touchevent->accept();
return true;
```

Для тех событий, которые не относятся к тачскрину, вызывается обработчик класса-родителя:

```
return MImageWidget::event(event);
```

Каждое событие тачскрина содержит список точек touchPoints(), обычно состоящий из одной точки, которые мы будем использовать для рисования на изображении – просто соединяя предыдущую и текущую точку линией. В приложении это происходит прямо в обработчике событий – виджет выдает текущий `QPixmap`, на который с помощью `QPainter` наносится линия и обновленный `QPixmap` устанавливается виджету. Конечно, это не самое оптимальное с точки зрения производительности и организации программы решение. Правильнее было бы вынести из обработчика все ресурсоемкие операции, такие как обновление изображения, или вообще перенести их в специальный обработчик paintEvent(), отвечающий за отрисовку графических виджетов. Чтобы сэкономить ресурсы, отрисовка производится не для каждой следующей точки, а только в случае, если расстояние между точками превышает определенный порог (или в случае, когда эта точка последняя).

К сожалению, несмотря на перехват событий тачскрина в `MyWidget`, они все равно продолжают попадать в компоновщик в виде `gestures`, и одновременно с рисованием на виджете происходит его движение внутри главного окна приложения. Чтобы избавится от этого нежелательного эффекта, необходимо также перехватить события `gesture`, для этого надо использовать метод

```
grabGesture(Qt::PanGesture);
```

В уже созданном обработчике `bool MyWidget::event(QEvent *event)` надо добавить перехват события `QEvent::Gesture`, с ним ничего не надо делать по существу, просто отметить как обработанное, чтобы оно не попало к компоновщику.

В конце работы изображение может быть сохранено в файл, для этого в *Qt* есть специальный класс `QImageWriter`. Он автоматически определяет формат сохраняемого файла по заданному расширению.

Перед сохранением `QPixmap`, полученный от виджета, необходимо сконвертировать в `QImage`. Все это реализовано в слоте `saveImage()`.

Вернемся к захвату видеокадра, который будет служить фоном для рисунка. Захват реализован в виде отдельного приложения `v4l2grab`, работающего напрямую с видеоподсистемой ядра MeeGo, которое можно собрать непосредственно на планшете, без использования *Qt* Creator. Установите зависимости

```
zypper install libjpeg-devel
```

скомпилируйте исполняемый файл

```
gcc v4l2grab.c -o v4l2grab -ljpeg
```

и перенесите его в /usr/bin

```
mv v4l2grab /usr/bin
```

Теперь v4l2grab можно использовать в главном приложении с помощью класса `QProcess`, который позволяет запускать внешние приложения из *Qt*. Целесообразно установить таймаут выполнения внешнего приложения, в данном случае это 3000 миллисекунд, т.е. 3 секунды. Созданное v4l2grab изображение сохраняется в файл `/tmp/webcam_image.jpg`, откуда потом его читает `QImageReader`:

```
v4l2grab.start("v4l2grab -W 800 -H 600 -d
/dev/video0"
" -o /tmp/webcam_image.jpg");
v4l2grab.waitForFinished(3000);
QImageReader jpg("/tmp/webcam_image.jpg");
jpg.read(&image);
```

Фоновое изображение также можно создать программно, для этого используется уже упомянутый `QPainter`, обладающий богатым набором графических примитивов.

Для удобства пользователя приложение, помимо виджета с изображением, оснащено тремя кнопками с помощью компоновщика и механизма слотов и сигналов.

10.3. Задания для самостоятельной работы

1. Добавить возможность изменять параметры пера (цвет, толщину, ...) при рисовании
2. Добавить диалоги для открытия файла изображения с диска и записи на диск
3. Использовать вместо утилиты `v4l2grab` компоненту Multimedia библиотеки *Qt* Mobility.

Список литературы

1. Gestures Programming. http://doc.qt.nokia.com/4.7-snapshot/gestures-overview.html
2. Qt Mobility 1.2 Multimedia API. http://doc.qt.nokia.com/qt-mobility-snapshot/multimedia.html