Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Курсовая работа на тему:

“Разработка трехмерного игрового движка для игры под Android”

Зольников Павел, 361 гр.

Научный руководитель:

Валентин Оносовский, исполнительный директор Exigen Services,

выпускник кафедры системного программирования 1985 г.

Санкт-Петербург

2011

Оглавление

Введение

Платформа Android

Android — операционная система для мобильных телефонов, основанная на ядре Linux. Эта система довольно молода, первое устройство (HTC T-Mobile G1) с Android было презентовано в сентябре 2008 года. Для сравнения: первое устройство с WinMobile – арпель2000, Symbian Ltd была основана в 1998, а первые смартфоны с этой системой на борту появились также в 2000 году. Вернемся к Android: изначально эта система разрабатывалась компанией Android Inc., которую затем купила Google. Впоследствии Google инициировала создание бизнес-альянса Open Handset Alliance (OHA), который сейчас и занимается поддержкой и дальнейшим развитием платформы. На данный момент в OHA входит 65 компаний, включающих в себя производителей мобильных телефонов, разработчиков программного обеспечения, некоторых мобильных поставщиков и изготовителей чипов. Среди них: Google, HTC, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia. Платформа Android уже очень популярна на рынке мобильных устройств Соединенных Штатов. Так, согласно данным исследовательской конторы NPD Group, в первом квартале 2010 года Android OS обошла iPhone OS по популярности. По данным NPD на сегодняшний день у Google 28% рынка, что ставит их на второе место после Research in Motion (36%), iPhone OS находится на третьем месте с 21%.

Android используется не только, как мобильная операционная система. Например, шведская компания People of Lava анонсировала телевизор на платформе Android. Также на Mobile World Congress 2010 было анонсировано несколько планшетов и нетбуков с Android в качестве операционной системы, созданы роботы, управляющим устройством которых являются смартфоны с Android OS. Так же на Google IO 2010 было анонсировано Google TV, основанное тоже на Android OS.

Разработка под Андроид

Приложения для Android являются программами в нестандартном байт-коде для виртуальной машины Dalvik. Dalvik Virtual Machine — основанная на регистрах виртуальная машина, она оптимизирована для низкого потребления памяти. Программы для Dalvik пишутся на языке Java. Несмотря на это, стандартный байт-код Java не используется, вместо него Dalvik VM исполняет байткод собственного формата. После компиляции исходных текстов программы на Java (при помощи javac) утилита dx из «Android SDK» преобразует .class файлы в формат .dex (Подробное описание dex формата http://www.dalvikvm.com/), пригодный для интерпретации в Dalvik. В версии

Android 1.5 разработчики добавили Native Development Kit, который позволяет писать собственные низкоуровневые модули для системы на языке C/C++, опираясь на стандартные linux-библиотеки.

OpenGL ES

OpenGL ES (OpenGL for Embedded Systems — OpenGL для встраиваемых систем) — подмножество графического интерфейса OpenGL, разработанное специально для встраиваемых систем — мобильных телефонов, карманных компьютеров, игровых консолей. OpenGL ES определяется и продвигается консорциумом Khronos Group, в который входят производители программного и аппаратного обеспечения, заинтересованные в открытом API для графики и мультимедиа.

В настоящее время существует уже несколько версий спецификации OpenGL ES. OpenGL ES 1.0 написана по спецификации OpenGL 1.3, OpenGL ES 1.1 определена относительно OpenGL 1.5, OpenGL ES 2.0 определена относительно спецификации OpenGL 2.0. Версии 1.0 и 1.1 имеют профили common и common lite. Common lite отличается тем что поддерживает только вычисления на числах с фиксированной десятичной точкой, в то время как common поддерживает также и вычисления с плавающей точкой.

Версии:

- OpenGL ES 1.0 был выбран в качестве официального 3D API в Symbian OS и для платформы Android.
- OpenGL ES 1.0 плюс некоторые возможности 2.0 и Cg поддерживаются в PlayStation 3 как один из доступных графических API.
- OpenGL ES 1.1 используется в качестве графической библиотеки в iPhone SDK.
- OpenGL ES 2.0 используется в Nokia N900, поддерживается в Symbian, поддерживается в Android версии 2.2 и выше, используется в игровой консоли Pandora, а также в iPhone SDK 3.0 (только для iPhone 3GS и новых iPod Touch), поддерживается в Bada OS. Эти устройства также выбраны для использования WebGL, OpenGL для браузеров.

Постановка задачи

Общая часть

Целью проекта была разработка игры под платформу Android, в которой объект (в нашем случае шарик) перемещался бы в трехмерном лабиринте. Лабиринт автоматически генерируется по растровому изображению (карте), которую пользователь может нарисовать сам в графическом редакторе. Более того, присутствует возможность генерации нескольких уровней по нескольким картам. Также в игру встроен свой трехмерный пиксельный физический движок, а управление объектом происходит через акселерометр.

Моя часть

В рамках всего проекта передо мной ставилась задача по разработке трехмерного игрового движка для игры, включающего в себя действия по:

- Загрузке и отрисовке уровней и управляемого объекта
- Управлению объектом
- Освещению уровня

Также со временем появилась задача по оптимизации отрисовки уровня, нужная для того, чтобы снизить нагрузку на процессор.

Средства разработки

Разработка игры велась в среде Eclipse с использованием плагина «Android Development Tools» (ADT), предназначенного для Eclipse версий 3.3-3.5. Для работы с графикой использовались библиотеки Khronos OpenGL ES (стандарт OpenGL для мобильных устройств) и Android OpenGL.

Решение поставленных задач

Архитектура движка. Управление объектом

Архитектура моего движка основана на шаблоне проектирования Model-View-Controller. Моделью (Model) в данном случае является управляемый объект (шарик). В классе Ball есть поле acceleration, которое определяет ускорение шарика в данный момент. Это ускорение задает контроллер (Controller). Класс Controller реализует стандартный интерфейс SensorEventListener. При помощи переопределенного метода onSensorChanged() Controller получает новые данные с акселерометра и записывает их экземпляр класса Model. Когда ускорение шарика получено, начинается высчитывание следующего его положения. Это происходит с использованием движения, основанного на времени (time-based movement). Такое движение нужно, чтобы игра работала с одинаковой скоростью на устройствах разной мощности. После расчета следующего положения шарика происходит отрисовка всей сцены. Для этого используется последняя компонента архитектуры – представление (View). Она состоит из двух классов: GameSurfaceView (наследника *android.opengl.GL SurfaceView*) и SceneRenderer (реализации *android.opengl.GLSurfaceView.Renderer*). Подробнее об отрисовке будет рассказано ниже.

Загрузка уровня

Генерацией трехмерных объектов по двумерному изображению занималась Юля. Мне же нужно было использовать ее классы и объединить их, чтобы получить уровень. Для этого мной был создан класс Level, содержащий методы init(), loadTextures() и draw(). Первые два метода вызываются последовательно в конструкторе класса. В конструктор класса передаются:

- имена файлов с изображением уровня и текстурами
- экземпляр класса *javax.microedition.khronos.opengles.GL10* для того, чтобы загрузить текстуры
- контекст приложения для доступа к ресурсам, таким как изображение уровня и файл с текстурой

Метод init() служит для того, чтобы по двумерному массиву, созданному из изображения заполнить четыре буфера:

- vertexBuffer, содержащий координаты всех вершин на уровне
- texBuffer, содержащий тектурные координаты
- indexBuffer, содержащий номера вершин и определяющий, в каком порядке отрисовывать примитивы (треугольники)

- normalBuffer, содержащий нормали к вершинам

Метод loadTextures() принимает имя файла с текстурой, контекст приложения для доступа к этому файлу и экземпляр класса *javax.microedition.khronos.opengles.GL10* для того, чтобы эту текстуру загрузить.

Метод draw() занимается тем, что отрисовывает уровень на сцене. Отрисовка происходит за счет вызова следующих методов:

`gl.glEnableClientState(GL10.GL_VERTEX_ARRAY)` – включение работы с буфером вершин.

`gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer)` – установка начала массива вершин. Здесь определяется то, сколько компонент будет содержать вершина, а также смещение относительно начала массива вершин.

Далее происходит все то же самое, но уже с массивами текстурных координат и нормалей:

```
gl.glEnable(GL10.GL_TEXTURE_2D)
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY)
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texBuffer)
gl.glEnableClientState(GL10.GL_NORMAL_ARRAY)
gl.glNormalPointer(GL10.GL_FLOAT, 0, normalBuffer)
```

Последний метод отрисовывает наш уровень. Здесь определяется то, как будет происходить отрисовка(в нашем случае треугольниками), длина массива индексов, тип данных массива и указатель на начало массива:

```
gl.glDrawElements(GL10.GL_TRIANGLES,indices.length,
GL10.GL_UNSIGNED_SHORT, indexBuffer);
```

Загрузка управляемого объекта

Для того чтобы загрузить управляемый объект, нужно было найти парсер файлов в формате obj. Формат файлов OBJ — это простой формат данных, который содержит только 3D геометрию, а именно, позицию каждой вершины, связь координат текстуры с вершиной, нормаль для каждой вершины, а также параметры, которые создают полигоны. Было перепробовано несколько вариантов, и я в итоге остановился на библиотеке objloaderforandroid (http://sourceforge.net/projects/objloaderforand/). Для загрузки модели нужно создать экземпляры двух классов: OBJParser и TDModel. OBJParser при помощи метода parseObj разбирает входной .obj файл и сохраняет полученную по нему модель в экземпляр класса TDModel. В классе TDModel уже определен метод draw(), поэтому все, что потом останется – это вызвать этот метод из класса, занимающегося отрисовкой сцены.

Отрисовка сцены

Для того, чтобы отрисовать сцену, был создан класс SceneRenderer, реализующий интерфейс *android.opengl.GLSurfaceView.Renderer.* Были реализованы следующие методы.

Метод onSurfaceCreated() вызывается один раз при создании сцены, в нем определяются начальные параметры сцены. Мной в этот метод были добавлены вызовы методов для генерации уровня, загрузке управляемого объекта, а также вызов метода initLighting(), включающего и задающего параметры освещения сцены.

Метод onSurfaceChanged() вызывается всякий раз, когда меняются размеры окна (например, при повороте телефона). В него передаются новые размерности окна, тип рабочей матрицы (матрицы, на которую умножаются все координаты при преобразованиях) меняется на проективную, и происходит изменение окна.

И, наконец, метод onDrawFrame(), который вызывается каждый раз при отрисовке нового кадра. Здесь важно следующее. Для того чтобы объект двигался, необходимо произвести параллальный перенос всех его вершин. Здесь эту роль на себя берет метод glTranslatef(), который из нашей рабочей матрицы делает матрицу переноса на определенный вектор. Этот вектор берется из переменной nextFrame, определяющей следующее положение шарика. О том, как высчитывается этот вектор, было рассказано выше. Но проблема в том, что при вызове метода glTranslatef() на матрицу переноса умножается не только наш объект, но и вся сцена. Таким образом, если ничего не предпринять, вместе с объектом будет двигаться вся сцена. Чтобы избежать этого, необходимо до отрисовки объекта сохранить текущую матрицу, поместив ее на вершину стека, затем умножить на матрицу переноса вершины объекта (здесь наша рабочая матрица уже изменена) и отрисовать объект; после этого мы достаем старую

матрицу с вершины стека и отрисовываем оставшуюся сцену. Для того чтобы положить и снять матрицу с вершины стека, используются методы glPushMatrix() и glPopMatrix(), а процесс отрисовки объекта и сцены выглядит следующим образом:

```
gl.glPushMatrix();
gl.glTranslatef(nextFrame.getX(), nextFrame.getY(), nextFrame.getZ());
ball.draw(gl);
gl.glPopMatrix();
level.draw(gl);
```

Освещение сцены

Освещение сцены – важная часть, поскольку без него все объекты будут казаться плоскими. OpenGL предоставляет богатые возможности по созданию и настройке освещения. Для задания освещения был создан метод initLighting() в классе SceneRenderer. Он вызывается единственный раз из метода onSurfaceCreated() при создании сцены. Освещение включается при помощи вызова метода glEnable(GL10.GL_LIGHTING). В OpenGL изначально заложено восемь источников света, но для игры хватит и одного. Он включается при вызове метода glEnable(GL10.GL_LIGHT0). Как можно видеть, это источник с номером ноль. У источника света есть следующие параметры:

- ambient – параметр фонового освещения, если задать только его, то объекты не будут казаться трехмерными
- diffuse – этот параметр направленного освещения, его нужно обязательно задавать, чтобы объекты виделись трехмерными
- position – позиция источника света, обычно достаточно далеко вверху и не ровно над уровнем
- direction – направление освещения, если его не задать, то получим свет, который бьет во все стороны
- attenuation – угасание света, чем дальше объект от источника света, тем менее он освещен

Параметры задаются при вызове метода glLightfv(). В него передаются источник света, который мы настраиваем, название параметра и значение этого параметра. Для позиции и направления – это соответственно координаты и вектор, для ambient и diffuse – цвет в формате RGB, для угасания – тип угасания (линейное, квадратичное) и множитель этого угасания.

Оптимизация отрисовки

Уровень в игре состоит из большого (тысячи) числа вершин и нормалей. Однако далеко не все они попадают в область видимости, так как уровни достаточно большие. Несмотря на это, все вершины уровня подаются на обработку процессору, но отрисовывается только часть из них. Следовательно, процессор нагружается ненужной работой. К счастью, есть хорошее решение этой проблемы. Весь уровень разбивается на блоки, и на обработку процессору подаются только те из них, вершины которых попали в область видимости. Хотя сам по себе процесс разбиения уровня на блоки не так прост, в случае нашей игры все становится значительно проще. Дело в том, что изначально у нас есть растровое изображение – карта, по которой потом будет генерироваться уровень. Поэтому сначала можно разбить ее на блоки, а потом по этим блокам генерировать части уровня.

Заключение

В рамках данной курсовой работы было выполнено следующее:

- Реализована загрузка уровня и объекта на сцену с их последующей отрисовкой
- Реализовано управление на акселерометре
- Реализованы задание и настройка освещения сцены
- Оптимизирована отрисовка уровня

Список использованных источников

1. http://www.opengl.org/
2. http://developer.android.com/
3. Mike Smithwick, Mayank Verma. Pro OpenGL ES for Android, Apress, 2012
4. M. Woo, T. Davis, J. Neider, D. Shreiner. OpenGL, Programming Guide, 4th edition, 2006
5. Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели, Диалог-МИФИ, 2001