Июль 2005

Подарок от Redikod:
Учебное пособие по 3D- программированию для мобильных устройств, используя M3G (JSR 184)

Это – первая часть из запланированной серии учебных пособий по Mobile Java 3D, созданных Mikael Baros, старшим программистом из Redikod, поможет Вам начать разработку на Mobile Java 3D.

Mikael – регулярный докладчик и полемист в дискуссиях на форуме разработчиков Sony Ericsson, и Вы, возможно, болтали с Mikael под его псевдонимом "Biovenger".

Redikod, из Мальмо(Malmo) в Швеции, - разработчик с 1997 сетевых и мобильных игр, и эта маленькая компания - теперь один из лидеров в скандинавской промышленности игр. Его непропорциональное влияние происходит от стратегических инициатив типа Скандинавского Потенциала Игр, ежегодной конференции, и скандинавского участия в E3 2006. Redikod уполномочен проектировать скандинавскую общественную систему поддержки и финансирования разработок для развития игр, включая мобильный телефон, что, ожидаемое заключительное принятие этой системы произойдет этой осенью и войдет в силу в 2006. Но разработка 3D- и мульти-плеерных для мобильного телефона - их ежедневная работа.

И так, первый взнос от Mikael-a.

Часть первая: Быстрый скачок в мир программирования Mobile Java 3D

Введение

Для начала я хотел бы, чтобы Вы знали о нескольких Веб- адресах, которые будут очень полезны в вашей поездке по стране M3G.
Прежде всего, и вероятно наиболее важно, является секция посвященная Mobile Java 3D на портале Мир Разработчиков фирмы Sony Ericsson: Sony Ericsson Developer World.
Во вторых, если у Вас когда-либо будут проблемы –посетите форум Sony Ericsson Mobile Java 3D forum. Для всего остального, используйте Web портал Мир Разработчиков Sony Ericsson, где Вы найдете ответы на ваши вопросы и более того.

Теперь, когда Вы знаете, куда идти, если есть проблемы, давайте продолжим процесс обучения. Цель этой обучающей программы состоит в том, чтобы преподать Вам, как настроить ваш собственный 3D-Холст (Canvas) и рендерить на экране. Чтобы рендерить модели, я сначала покажу Вам, как их загрузить и расскажу об инструментах, которые являются доступными, чтобы создать модели M3G. Мы закончим, управлением камерой так, чтобы мы могли побродить по нашей сцене.

Я только хочу, чтобы Вы почувствовали, как быстро можно разработать 3D-приложение с M3G, так что эта обучающая программа будет довольно быстрая и прямая с неглубоким объяснением. Другие части этой серии подробно исследуют различные темы M3G.

Так как код предназначен для образовательных целей, он не оптимален, и при этом не обрабатывает все возможные ошибки. Они будут рассмотрены в более продвинутых темах, к которым будем обращаться позже.

Что Вы должны знать

Прежде, чем Вы начнете это читать, Вы должны знать основы MIDlet класса и класса Холста. Это - не обязательное требование и если Вы почувствуете себя неуверенным, смотрите исходный код (распределенный с этим учебным пособием), и проверьте классы M3GCanvas и M3GMIDlet. Будет очень хорошо, если Вы имеете некоторую базу в математике 3D-программирования, но это не обязательно.

Холст

Когда мы разрабатываем в JSR 184, мы будем использовать профиль MIDP 2.0, что означает, что мы свободны в использовании несколько больше функций. Давайте начнем с настраивания нашего Холста. Это - та же самая процедура как с обычной 2D- Java игрой, Вы стартуете ваш MIDlet класс,

создаете ваш Холст и чертите в вашем методе paint. Это - довольно простой процесс и так как Вы уже должны это знать, я буду только вскользь упоминать это. Давайте сначала рассмотрим начало класса Холста: импорт и декларациию переменных.

```
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.M3G.Camera;
import javax.microedition.M3G.Graphics3D;
import javax.microedition.M3G.Light;
import javax.microedition.M3G.Loader;
import javax.microedition.M3G.Object3D;
import javax.microedition.M3G.Transform;
import javax.microedition.M3G.World;

/**
 *
 * @author  Biovenger
 * @version
 */
public class M3GCanvas extends GameCanvas implements Runnable {


    // Управление нитью(Thread)l
    boolean running = false;
    boolean done = true;

    // Если игра должна закончиться
    public static boolean gameOver = false;

    // Намеки рендеринга(Rendering hints;Параметры рендеринга)
    public static final int STRONG_RENDERING_HINTS = Graphics3D.ANTIALIAS |
Graphics3D.TRUE_COLOR | Graphics3D.DITHER;
    public static final int WEAK_RENDERING_HINTS = 0;
    public static int RENDERING_HINTS = STRONG_RENDERING_HINTS;

    // Массив клавиш
    boolean[] key = new boolean[5];

    // Константы клавиш
    public static final int FIRE = 0;
    public static final int UP = FIRE + 1;
    public static final int DOWN = UP + 1;
    public static final int LEFT = DOWN + 1;
    public static final int RIGHT = LEFT + 1;
```

Это общий материал, но давайте, быстро его рассмотрим и продолжим. Прежде всего мы мпортируем все классы, которые мы собираемся использовать в этой обучающей программе, и Вы можете найти их в документации по API JSR 184. Мы также определяем некоторые переменные, типа переменной управления нитью процесса и сделанные так, что довольно очевидно их назначение.

Теперь, давайте рассмотрим желаемые намеки(hints; параметры) рендеринга. Эти "намеки" - режимы сказать мобильному устройству о том, какое Вы хотите иметь качество рендеринга. Однако, так как они - намеки, не гарантируется, что мобильное устройство будет действовать по ним. Здесь я определяю два различных параметра. Слабый и Сильный. Как Вы видите, Сильные намеки содержат и сглаживание (anti-aliasing), истинный цвет (true color) и возбуждение(dithering). Слабый - не содержит никаких намеков вообще, и является в основном намеком плохого качества рендеринга но быстрого. Вы видите в коде, что намеки могут быть объединены простым логическим ИЛИ. Я буду говорить больше о намеках в более поздней части этой серии обучающей программы.

Затем мы определяем массив клавиш, это - очень простой массив, который содержит логическое True если клавиша была нажата. Если Вам интересно как обрабатываются нажатие клавиш, посмотрите исходный код этого примера. Будьте уверены, чтобы сказать: если (key[]) равен Вы True, то только что нажат ключ UP.

Формат M3G файла

Стандарт JSR 184 имеет собственный формат файла, названный M3G. Этот очень универсальный 3D- формат может держать много данных типа моделей, огней, камер, текстур и даже анимаций. Остро! Не только формат, действительно хороший, но и очень легко загрузить его в ваше приложение, хотя об этом позже.
Во всяком случае, я держу пари, что Вы думаете "M3G? Никогда не слышал об этом. Как я создам M3G файл? " и даже если Вы не знаете это, я объясню. Есть многочисленные режимы создать M3G файлы:

1. Прежде всего, последняя редакция 3D Studio Max от Discreet имеет встроенный экспортер в M3G. Только нажмите кнопку Export, и Вы можете экспортировать вашу полную сцену, анимацию, кости, материалы и все, в M3G файл. Однако, многие считают экспортер от Discreet немного тяжелым, и имеющим немного ошибок, так что для лучших результатов, используйте метод 2.

2. HiCorp, являющаяся также поставщиком реализации(микрокода) JSR 184 в телефонах Sony Ericsson's, создала очень мощные экспортеры для 3 самых популярных 3D- программ моделирования. Они доступны для 3D Studio Max, LightWave and Maya. Вы можете найти их здесь>>.

3. Blender, мощный и свободный инструмент 3D- моделирования также имеет в наличии экспортера в M3G. Однако, он является все еще в ранней разработке и имеет небольшую функциональность. Проверьте Blender здесь>>.

Так, как мы загружаем эти очень мощные файлы в нашу программу? Очень легко. JSR 184 содержит класс Лоадер(Loader), и он просто загружает файлы. Вызовом одного простого метода, можно загрузить все ссылки из M3G файла. Метод Loader.load имеет два различных списка параметров. Каждый берет URL как строку(String),а другой вдобавок и массив байтов низкого уровня(raw). Вот - пример того, как это используется.

```
Object3D[] objects = Loader.load("file.M3G");
Object3D[] objects2 = Loader.load(byteArray, offset);
```

Метод загрузки всегда возвращает массив объектов Object3Ds и есть очень серьезное основание для этого. Лучшее - то, что класс Лоадера может загрузить намного больше, чем только M3G файлы, он может загрузить сохраненный(serialize) любой класс, который наследован от Object3D. Однако, Вы будете главным образом использовать это для того, чтобы загрузить M3G файлы.

Теперь, я создал простой M3G файл, названный M3G-картой и я хочу рендерить его. Чтобы загружать этот файл, мы будем использовать метод Loader.load, однако Вы только что видели, он возвращает массив Object3D. Мы не можем использовать массив Object3D, чтобы сразу рендерить, поскольку должны это конвертировать в то, что мы можем рендерить на экран позже. В этой обучающей программе мы загрузим узел Мира(World node). Узел Мира - главный узел графа сцены в JSR 184. Он содержит все виды информации: камеры, освещение, фон и большое количество мешей(meshes). Я освещу графы сцены и реализацию в JSR 184 графов сцены в более поздней части этой серии, пока Вы только должны знать, что класс Мира может содержать целую сцену, и это - то, что мы хотим! Проверьте этот метод, он загружает узел Мира из M3G файла.

```
/ ** Загружаем наш мир */
   private void loadWorld()
   {
      try
      {
         // Загрузка мира очень проста. Обратите внимание, что я люблю использовать а
         // папку ресурсов проекта(res-folder), где я содержу все файлы.
         // Если Вы обычно помещаете ваши
         // ресурсы только в корне проекта, то загрузите их из корня.
           Object3D[] buffer = Loader.load("/res/map.M3G");

         // Найти узел Мира, лучше всего сделать это "безопасным" путем
         for(int i = 0; i < buffer.length; i++)
         {
            if(buffer[i] instanceof World)
            {
```

```
                world = (World)buffer[i];
                break;
            }
        }

        // Очистка объектов
        buffer = null;
    }
    catch(Exception e)
    {
        // ОШИБКА!
        System.out.println("Loading error!");
        reportException(e);
    }
}
```

Как Вы можете видеть, после того, как мы загружаем массив Object3D классом Лоадера, мы просто проходим полный массив и ищем порождение от класса Мир. Это - самый безопасный режим найти узел Мира. После того, как найдем, мы только прерываем цикл и чистим наш буфер (Это, не обязательно, поскольку он был бы автоматически очищен после завершения метода(loadWorld()). Хотя это хороший стиль программирования. Прим. Уменьшает время на автоматическую борьбу с утечкой памяти).

Хорошо, теперь мы загрузили наш узел Мира, который как я уже Вам сказал - главный узел графа сцены и содержит всю информацию сцены.Прежде, чем я покажу Вам как легко это отрендерить, давайте сначала извлечем и настроим камеру из мира, чтобы мы могли двигаться, в загруженном мире.

Управление камерой

Мы имеем наш узел Мира, готовый к рендерингу и теперь все в чем мы нуждаемся –это в камере, которую мы можем перемещать в мире. Если Вы помните, я уже сказал Вам, что узел Мира содержит информацию Камеры, так что мы должны быть режимны извлечь камеру из Мира и управлять ей.

Камера в JSR 184 описана классом Camera. Этот класс делает очень легким управление камерой в нашем 3D- приложении простым перемещением и методами ориентации. Единственные два метода, которые мы будем использовать в этом примере translate(float, float, float) и setOrientation(float, float, float, float). Первый просто перемещает камеру в 3D- пространстве на смещения x, y и z. Так, например, если бы Вы хотели переместить камеру на 3 единицы по Оси X и 3 единицы по Оси Z, Вы сделали бы так:

```
Camera cam = new Camera();  // Это наша камера
//Перемещение камеры   X  Y  Z
cam.translate(3.0f, 0.0f, 3.0f);
```

Часть пирога! Каждый вызов метода переводит камеру далее, так что два вызова вышеупомянутых методов фактически перевели бы камеру 6 единиц по оси X и 6 единицам по оси Z. Вращение столь же легко, но я должен объяснить сначала этот метод. Это работает подобно почти всем 3D- методам вращения API. Вы имеете четыре параметра, первый - фактическое вращение в градусах, и последние три составляют вектор ориентации (xAxis, yAxis, zAxis), для оси вращения. Ориентация и векторы ориентации рассмотрим позже в серии статей, пока, только знайте это:

```
// Поворот камеры на 30 градусов вокруг оси X
cam.setOrientation(30.0f, 1.0f, 0.0f, 0.0f);
// Поворот камеры на 30 градусов вокруг оси Y
cam.setOrientation(30.0f, 0.0f, 1.0f, 0.0f);
// Поворот камеры на 30 градусов вокруг оси Z
cam.setOrientation(30.0f, 0.0f, 0.0f, 1.0f);
```

Обратите внимание, что метод называют setOrientation, что означает, что он фактически очищает любое предыдущее вращение, которое Вы, возможно, сделали. Я предположу, что Вы уже знаете то, что означает вращение вокруг оси и не входит в детальное рассмотрение этого параграфа.

Теперь, когда Вы знаете, что заставляет камеру двигаться и вращаться, Вы желаете, чтобы я показал Вам, как извлечь Камеру из Мира.

```
/** Загрузка нашей камеры */
   private void loadCamera()
   {
      // Плохо! Нет Мира
      if(world == null)
         return;

      // Получить активную камеру от мира
      cam = world.getActiveCamera();

      // Создать свет
      Light l = new Light();

      // Сделаем свет  ОКРУЖАЮЩИМ
      l.setMode(Light.AMBIENT);

      // Мы хотим немного более высокую интенсивность
      l.setIntensity(3.0f);

      // Добавить свет к нашему миру
      world.addChild(l);
   }
```

Это - это просто? Да, это что просто. Мы только извлекаем активную камеру из Мира, используя getActiveCamera. Это дает нам камеру, с которой мир экспортировался.С этим методом(loadCamera()) мы имеем камеру, которую мы можем перемещать вокруг настолько, насколько мы желаем. Однако, метод делает кое-что еще, он добавляет свет! Мы будем копаться глубже в источниках освещения в более поздних частях, но здесь Вы видите, как легко добавить свет к вашему миру. Я создаю Окружающий(Ambient) свет (для Вас, кто не знает, окружающий свет освещает все поверхности по всем направлениям) и добавляю его к миру. Этим путем мы получаем очень хорошо освещенный мир.
Поскольку ранее я сказал, что узел Мира можете держать все виды информации, включая свет, так что мы только должны добавить свет к нашему миру и JSR 184 сделает остальное за нас. Это не удобно?
Прежде, чем мы доберемся к последней части; рендерингу, давайте сделаем движение камеры. Я же сказал Вам, что массив логических значений нажатий клавиш, содержит информацию о нажатых клавишах, поэтому все, что мы должны сделать – это проверять массив и заставлять нашу камеру вести себя в зависимости от проверок. Прежде всего, мы будем нуждаться в некоторых переменных, чтобы заставить нашу камеру повиноваться.

```
// Вращение камеры
float camRot = 0.0f;
double camSine = 0.0f;
double camCosine = 0.0f;

// Подпрыгивание камеры вверх
float headDeg = 0.0f;
```

Эти переменные помогут нам держать след от вращения камеры, тригонометрию и подпрыгивание камеры вверх. Тригонометрия используется для движения позже. Подпрыгивание камеры весьма просто, это - только дешевый эффект, который мы вставим для более натурального чувства, чтобы заставить камеру подпрыгивать, поскольку мы идем в мире. Все хорошо, все что мы должны сделать – это движение камеры. Это сделано в следующем методе:

```
private void moveCamera() {
      // Проверка клавиш
      if(key[LEFT])
```

```
        {
            camRot += 5.0f;
        }
        else if(key[RIGHT])
        {
            camRot -= 5.0f;
        }

        // Установка вращения
        cam.setOrientation(camRot, 0.0f, 1.0f, 0.0f);

        // Вычисление тригонометрии для перемещения камеры
        double rads = Math.toRadians(camRot);
        camSine = Math.sin(rads);
        camCosine = Math.cos(rads);
```

Как Вы можете видеть, что эта половина метода довольно проста. Сначала мы проверяем, нажал ли пользователь LEFT или RIGHT клавишу джойстика и если так, то мы только увеличиваем или уменьшаем вращение камеры. Это достаточно просто.
Следующие несколько строк интересны. Мы хотим, чтобы голова повернулась, поскольку пользователь нажимает направо или налево, так что мы будем вращаться вокруг Оси Y, что означает вектор ориентации 0.0f, 1.0f, 0.0f. После того, как мы вращаем камеру, мы вычисляем новый Синус и Косинус угла, которые используются позже для вычисления движения. Теперь следующяя половина метода:

```
if(key[UP])
        {
            // Перемещение вперед
            cam.translate(-0.1f * (float)camSine, 0.0f, -0.1f * (float)camCosine);

            // Подпрыгивание вверх головы
            headDeg += 0.5f;

            // Простой путь "Подпрыгивания вверх" камеры,
            // когда она перемещается с пользователем
            cam.translate(0.0f, (float)Math.sin(headDeg) / 40.0f, 0.0f);
        }
        else if(key[DOWN])
        {
            // Перемещение назад
            cam.translate(0.1f * (float)camSine, 0.0f, 0.1f * (float)camCosine);

            // Подпрыгивание вниз головы
            headDeg -= 0.5f;

           // Простой путь "Подпрыгивания вниз" камеры,
            // когда она перемещается с пользователем
            cam.translate(0.0f, (float)Math.sin(headDeg) / 40.0f, 0.0f);
        }
       // Если пользователь нажимает ключ FIRE , давайте выйдем из мидлета
        if(key[FIRE])
            M3GMidlet.die();
    }
```

Здесь мы проверяем для UP или DOWN. UP переместит камеру вперед ,а DOWN переместит назад. Они - действительно простые перемещения, но я их быстро объясню. Камера стартуя всегда начинает смотреть вниз по отрицательной Оси Z, так чтобы продвигать камеру, мы только должны переместить ее по отрицательной Оси Z.
Однако, после вращения камеры, мы не можем переместить ее только по Оси Z, это будет выглядеть неправильно. Мы должны переместить камеру и по Оси X также, чтобы мы получили движение, которое мы желаем. Это достигнуто, используя тригонометрические функции. Так как это - не обучающая программа по 3D- математике, я не буду вдаваться в подробности, в конце концов Вы

должны уже знать это, а если Вы находите это непонятным, то ищите хорошую 3D- математическую обучающую программу на Интернете.
После каждого перевода мы также подпрыгиваем. Я только передаю в метод трансляции функциюй синуса по Оси Y, так, чтобы это напомнило, что голова идет вверх и вниз, именно поэтому мы или увеличиваем или уменьшаем headDeg переменную каждый раз, когда камера перемещена. Мы также в конце проверяем нажатие клавиши FIRE, чтобы пользователь мог выйти из мидлета, когда он хочет. (Он может также использовать невидимую команду EXIT, которую я добавил(в мидлет), когда холст был создан).
Вот именно! Это - все наше продвинутое движение камеры, теперь все что осталось, это рендерить на рендеринг узел Мира!

Рендеринг

Прежде, чем мы впрыгнем в код, я должен рассказать Вам о непосредственном и сохраненном режимах рендеринга. Сохраненный режим - то, что мы используем в этой обучающей программе, и это в основной - режим, который Вы используете, когда Вы рендерите полный узел Мира со всеми его камерами, источниками света и мешами
Это - самый легкий режим, чтобы рендерить, но также и режим, Вы имеете наименьший контроль над вашим миром. Непосредственный режим - то, когда Вы непосредственно рендерите Группу мешей, одиночный мешь или данные вершин. Это дает Вам намного больший контроль, так как с каждым рендерингом, Вы передаете матрицу преобразования, которая преобразовывает объект перед рендерингом.Вы можете рендерить узел Мира в непосредственном режиме, указывая матрицу преобразования в вызове метода рендеринга, но тогда Вы игнорировали бы все эффекты узла Мира: камеры, фон и другие. Я буду детализировать о двух различных режимах рендеринга в более поздних статьях этой серии. Пока, давайте увидим, как мы можем рендерить мир.

Graphics3D

Весь рендеринг в JSR 184 делает объект Graphics3D. Он может даже содержать информацию о камере и освещении, если Вы рендерите в непосредственном режиме. Давайте сейчас не волноваться об этом, хотя это - проблема, к которой я обращусь позже.

Чтобы рендерить объектом Graphics3D, Вы сначала должны связать его с графическим контекстом. Графический контекст в основном означает любой Графический объект, который чертит что-то. Это мог быть Графический объект Изображения, если Вы хотите рендерить в Изображение, или это мог бы быть главный Графический объект, полученный от метода getGraphics ().
Используя главный Графический объект Вы рендерите непосредственно на экран, и это то, что нам надо. Получить объект Graphics3D просто, Вы только вызываете метод Graphics3D.getInstance (). Вы получаете только один объект Graphics3D на мидлет, который Вам станет доступным, как только получите его от метода getInstance(). Связывание сделано методом bindTarget, и есть несколько способов использовать его. Вот - несколько примеров.

```
//Здесь наш объект Graphics3D
Graphics3D g3d = Graphics3D.getInstance();

// Связать с изображением
Image img = Image.createImage("myImage.png");
Graphics g = img.getGraphics();
g3d.bindTarget(g);

// Связать с главным Графическим объектом
g3d.bindTarget(getGraphics());

// Мы можем также снабдить рендеринг намеками(hints;
// желаемыми режимами рендеринга).
// Помните те? Я говорил о них вначале.
// Это сделано, используя другую форму метода bindTarget.
```

```
// Требуется Графический объект для начала, как всегда,
//  затем требуется булевское значение и маска намеков.
// Булевское значение просто говорит объекту Graphics3D
// о необходимости использования буфера глубины.
// и Вы будете вероятно всегда устанавливать это в 'истину'(true).
// Здесь, мы будем использовать связку с нашими намеками:
g3d.bindTarget(getGraphics(), true, RENDERING_HINTS);
```

Теперь, когда Вы знаете, как связать вашу цель, Вы также должны знать, что цель должна быть отвязана(released) на каждом. Это означает, что, когда Вы все прорендерите, Вы должны отвязать(released)цель.
Есть иногда проблемы с отвязыванием и закреплением, так что большинство людей помещают шаг цикла игры в блок выполнения/ловли ошибок(try/catch) и вставляют releaseTarget(отвязывание) , в финальный(finally) блок. Это - так, как мы сделаем в этом примере.
Now, let's take a look at the rendering method. To render something you can use a variety of rendering methods, but we'll only be interested in one today, the render(World) method. Simple huh? Yeah, you only need to supply your world node and it'll render it for you. Let's take a look at what our game loop will look like:

Теперь, давайте посмотрим метод рендеринга. Чтобы рендерить что-то, Вы можете использовать разнообразие форм методов рендеринга, но нас сегодня интересует один - метод render(World). Просто ха? Да, Вы только должны снабдить(прим. Graphics3D) вашим узлом мира, и он для Вас его прорендерит. Давайте посмотрим, как выглядит наш шаг цикла игры:

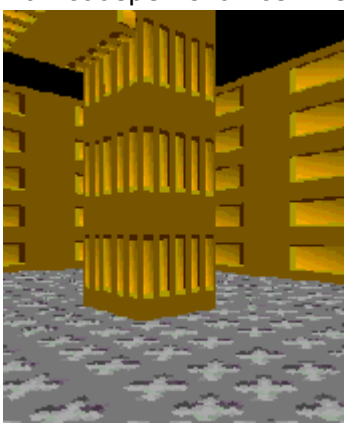
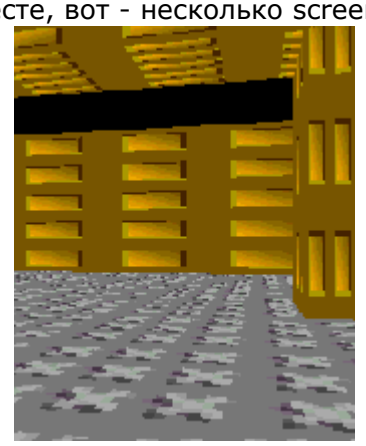
```
/** Чертим на экране
    */
   private void draw(Graphics g)
   {
      // Окутать все try/catch блоком на всякий случай
      try
      {
         //  Переместить камеру вокруг
         moveCamera();

         // Получить Graphics3D контекст
         g3d = Graphics3D.getInstance();

         // Сначала свяжите графический объект.
         // Мы используем наши предопределенные намеки рендеринга.
         g3d.bindTarget(g, true, RENDERING_HINTS);

         // Теперь, только отдайте мир. Просто как пирог!
         g3d.render(world);
      }
      catch(Exception e)
      {
         reportException(e);
      }
      finally
      {
         // Всегда не забудьте отвязать!
         g3d.releaseTarget();
      }
   }
```

Ничего себе, это - действительно короткий наш шаг цикла игры, давайте посмотрим, что он делает! Сначала вывается метод moveCamera, который перемещает и вращает нашу камеру. Мы видели это прежде.
Затем получаем контекст Graphics3D и связывает его с Графическим объектом, имеющим метод черчения(draw method). (Примечание: метод черчения вызывается методом выполнения(run) Нити (Thread's), которую порождает к этому времени глобальный Графический объект).Также добавлены намеки рендеринга, которые мы определили в начале нашего Холста. В конце концов все сделано, и

вызываем метод g3d.render (мир), который делает для нас всю работу. Он рендерит нашу полную сцену, петли, материалы, огни и камеру.

Заключение

Так соберем это все вместе, вот - несколько screenshots кода в действии:

Разве выглядит слишком потерто?

Так, вот - полный исходный код классов Canvas и MIDlet. В нем больше строк кода(прим. дополнительный код инфраструктуры выполнения мидлета) , по сравнению с кодом рендеринга на экран. Посмотрите код в начале обучающей программы, и увидите, как Canvas Холст работает внутренне. Или Вы можете загрузить это и играть с этим дома. Ниже Вы также найдете полный пакет JAR/JAD приложения.

M3GMidlet

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
public class M3GMidlet extends MIDlet implements CommandListener
{
   // Переменная, которая держит уникальный экран
 private Display display = null;

 // Холст(canvas)
 private M3GCanvas canvas = null;

 // Ссылка мидлета на самого себя
 private static MIDlet self = null;

/ ** Вызывается, когда приложение стартует, и когда возобновлено.
  * Мы игнорируем резюме(resume) здесь и помещаем данные для нашего
  *(прим. лучше бы в классе холста)
  * в startApp методе. Это - вообще то очень плохая практика.
  */

 protected void startApp() throws MIDletStateChangeException
 {
    // Выделение
  display = Display.getDisplay(this);
  canvas = new M3GCanvas(30);

   // Добавить к холсту команду выхода
  // Эта команда не будет видна так  как мы
  // работаем в полноэкранном режиме,
  // но всегда хорошо иметь команду выхода
```

```
 canvas.addCommand(new Command("Quit", Command.EXIT, 1));

 // Установка листенера мидлета
 canvas.setCommandListener(this);

 // Старт холста
 canvas.start();
 display.setCurrent(canvas);

 // Установка ссылки на самого себя
 self = this;
}
/** Вызывается, когда игра должна делать паузу */
protected void pauseApp()
{

}
/** Вызывается когда приложение должно быть закрыто */
protected void destroyApp(boolean unconditional) throws MIDletStateChangeException
{
   // Метод закрывает MIDlet
 notifyDestroyed();
}
/** Слушает команды и обрабатывает */
   public void commandAction(Command c, Displayable d) {
      // Если мы получаем команду  EXIT(ВЫХОД), мы уничтожаем приложение
      if(c.getCommandType() == Command.EXIT)
         notifyDestroyed();
   }

  / ** Статический метод, который выходит из приложения
    * используя статическую переменную  ' self' */
     public static void die()
   {
      self.notifyDestroyed();
   }
}
```

M3GCanvas

```
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.m3g.Camera;
import javax.microedition.m3g.Graphics3D;
import javax.microedition.m3g.Light;
import javax.microedition.m3g.Loader;
import javax.microedition.m3g.Object3D;
import javax.microedition.m3g.Transform;
import javax.microedition.m3g.World;
import javax.microedition.m3g.Mesh;

public class M3GCanvas extends GameCanvas implements Runnable {
   // управление нитью(Thread-control)
   boolean running = false;
   boolean done = true;

   // Если игра закончилась
   public static boolean gameOver = false;

   // намеки рендеринга
   public static final int STRONG_RENDERING_HINTS = Graphics3D.ANTIALIAS |
Graphics3D.TRUE_COLOR | Graphics3D.DITHER;
   public static final int WEAK_RENDERING_HINTS = 0;
   public static int RENDERING_HINTS = STRONG_RENDERING_HINTS;
```

```
// Массив клавиш
boolean[] key = new boolean[5];

// Константы клавиш
public static final int FIRE = 0;
public static final int UP = FIRE + 1;
public static final int DOWN = UP + 1;
public static final int LEFT = DOWN + 1;
public static final int RIGHT = LEFT + 1;

// Глобальная матрица идентичности
Transform identity = new Transform();

// Глобальный объект Graphics3D
Graphics3D g3d = null;

// Глобальный объект мира
World world = null;

// Глобальный объект камеры
Camera cam = null;

// Вращение камеры
float camRot = 0.0f;
double camSine = 0.0f;
double camCosine = 0.0f;

// Подпрыгивание (Удар в голову)
float headDeg = 0.0f;

/** Конструктор холста
 */
public M3GCanvas(int fps)
{
   // Мы не хотим захватить клавиши обычным путем
   super(true);

   // Мы желаем полноэкранный холст
   setFullScreenMode(true);

   // Загружаем мир
   loadWorld();

   // Загружаем камеру
   loadCamera();
}


 /** Когда установлен полноэкранный режим,
 * некоторые устройства будут вызывать
 * этот метод уведомляя нас о новой ширине/высоте.
 * Однако, мы в этой обучающей программе
 * действительно не заботимся о ширине/высоте
 * так что мы позволяем этому быть
 */

public void sizeChanged(int newWidth, int newHeight)
{

}

/** Загрузка камеры */
private void loadCamera()
```

```
  {
    // ПЛОХО!
    if(world == null)
      return;

    // Получить активную камеру от мира
    cam = world.getActiveCamera();

    // Создать свет
    Light l = new Light();

    // Сделать его ОКРУЖАЮЩИМ
    l.setMode(Light.AMBIENT);

    // Мы хотим немного более высокую интенсивность
    l.setIntensity(3.0f);

    // Добавить это к нашему миру
    world.addChild(l);
  }

  /** Загрузка мира */
  private void loadWorld()
  {
    try
    {
      // Загрузка мира очень проста. Обратите внимание, что я люблю использовать а
      // res-папка, что я удерживаю все файлы. Если Вы обычно только помещаете ваш
      // ресурсы в проектном корне, затем загрузите это от корня.
      Object3D[] buffer = Loader.load("/res/map.M3G");

      // Найти мировой узел, лучше всего сделать этот "безопасный" путь
      for(int i = 0; i < buffer.length; i++)
      {
        if(buffer[i] instanceof World)
        {
          world = (World)buffer[i];
          break;
        }
      }

      // Очистка объектов
      buffer = null;
    }
    catch(Exception e)
    {
      // ОШИБКА!
      System.out.println("Loading error!");
      reportException(e);
    }
  }

/** Чертим на экране
   */
  private void draw(Graphics g)
  {
    // Окутать все try/catch блоком на всякий случай
    try
    {
      // Переместить камеру вокруг
      moveCamera();

      // Получить Graphics3D контекст
```

```
        g3d = Graphics3D.getInstance();

       // Сначала свяжите графический объект.
       // Мы используем наши предопределенные намеки рендеринга.
     g3d.bindTarget(g, true, RENDERING_HINTS);

     // Теперь, только отдайте мир. Просто как пирог!
     g3d.render(world);
    }
    catch(Exception e)
    {
        reportException(e);
    }
    finally
    {
       // Всегда не забудьте отвязать!
        g3d.releaseTarget();
    }
}


/**
 *
 */
private void moveCamera() {
   // Проверка клавиш
    if(key[LEFT])
    {
        camRot += 5.0f;
    }
    else if(key[RIGHT])
    {
        camRot -= 5.0f;
    }

    // Установка вращения
    cam.setOrientation(camRot, 0.0f, 1.0f, 0.0f);

   // Вычисление тригонометрии для перемещения камеры
     double rads = Math.toRadians(camRot);
    camSine = Math.sin(rads);
    camCosine = Math.cos(rads);

    if(key[UP])
    {
       // Перемещение вперед
        cam.translate(-2.0f * (float)camSine, 0.0f, -2.0f * (float)camCosine);

        // Подпрыгивание головы вверх
        headDeg += 0.5f;

       // Простой путь "Подпрыгивания вверх" камеры,
        // когда она перемещается с пользователем
        cam.translate(0.0f, (float)Math.sin(headDeg) / 3.0f, 0.0f);
    }
    else if(key[DOWN])
    {
        // Перемещение назад

        cam.translate(2.0f * (float)camSine, 0.0f, 2.0f * (float)camCosine);

        // Подпрыгивание головы вниз
        headDeg -= 0.5f;
```

```
      // Простой путь "Подпрыгивания вверх" камеры,
       // когда она перемещается с пользователем
       cam.translate(0.0f, (float)Math.sin(headDeg) / 3.0f, 0.0f);
    }

    // Если пользователь нажимает ключ FIRE , давайте выйдем из мидлета
    if(key[FIRE])
       M3GMidlet.die();
 }

 /** Стартует холст, разжигая нить(thread)
  */
 public void start() {
    Thread myThread = new Thread(this);

    // Сделайте чтобы  мы знали, что мы запущены
    running = true;
    done = false;

    // Старт
    myThread.start();
 }

 /** Управляемый, управляется целой нитью. Также сохраняет постоянный FPS
  */
 public void run() {
    while(running) {
       try {
          // Вызываем метод process(вычисляем клавиши)
          process();

          // Чертим все
          draw(getGraphics());
          flushGraphics();

          // Спим для предотвращения starvation
          try{ Thread.sleep(30); } catch(Exception e) {}
       }
       catch(Exception e) {
          reportException(e);
       }
    }

    // Уведомление о завершении
    done = true;
 }

 /**
  * @param e
  */
 private void reportException(Exception e) {
    System.out.println(e.getMessage());
    System.out.println(e);
    e.printStackTrace();
 }



/** Пауза в игре
  */
 public void pause() {}

 /** Останавливает игру
```

```
    */
    public void stop() { running = false; }

    /** Процесс обработки клавиш
     */
    protected void process()
    {
        int keys = getKeyStates();

        if((keys & GameCanvas.FIRE_PRESSED) != 0)
            key[FIRE] = true;
        else
            key[FIRE] = false;

        if((keys & GameCanvas.UP_PRESSED) != 0)
            key[UP] = true;
        else
            key[UP] = false;

        if((keys & GameCanvas.DOWN_PRESSED) != 0)
            key[DOWN] = true;
        else
            key[DOWN] = false;

        if((keys & GameCanvas.LEFT_PRESSED) != 0)
            key[LEFT] = true;
        else
            key[LEFT] = false;

        if((keys & GameCanvas.RIGHT_PRESSED) != 0)
            key[RIGHT] = true;
        else
            key[RIGHT] = false;
    }

    /** Проверка запуска Нити
     */
    public boolean isRunning() { return running; }

    /** Проверка комплектности выполнения если нить финишировала
     */
    public boolean isDone() { return done; }
}
```

P.S.