Январь 2006

Подарок от Redikod: Учебное пособие по 3D- программированию для мобильных устройств, часть пять: Рендеринг карты высот ландшафта с использованием M3G

Здание основанное на предыдущих четырех частях обучающей программы " 3D- программирование для мобильных устройств, используя M3G (JSR 184) ", созданных Mikael Baros, старшим программистом из Redikod, теперь ведет Вас через основы предоставления ландшафта и карты высот.

Ниже Вы можете загрузить и zip файлы исходного кода и пакета приложения для части пять, так же можете освежить вашу память первыми четырьмя частями из учебного пособия.

- **Часть первая: Быстрый скачок в мир программирования Mobile Java 3D**
- **Часть два: Теория 3D освещения и ориентации**
- **Часть три: системы Частиц и непосредственный режим рендеринга**
- **Часть четыре: встроенное столкновение M3G, физика света и перспективы камеры**

Часть пять: Рендеринг карты высот(heightmap) ландшафта с использованием M3G

Пятая часть путеводителя от Mikael-a.

Введение

Добро пожаловать в пятую часть этого M3G ряда обучающей программы. Сегодня я покажу Вам очень простую технику, которая используется в почти всех 3D- играх (в той или другой форме). Карта высот (heightmaps).

Используя карту высот, проектировщики/разработчики могут легко и быстро создать естественный ландшафт (возможно даже, используя perlin шумовой генератор, но не обязательно требующийся. Красота карты высот состоит в том, что вместо требуемого сложного понятия, типа красивого и реалистического 3D- ландшафта, и упрощает эту проблему и сводит ее к легкому 2D- изображению.

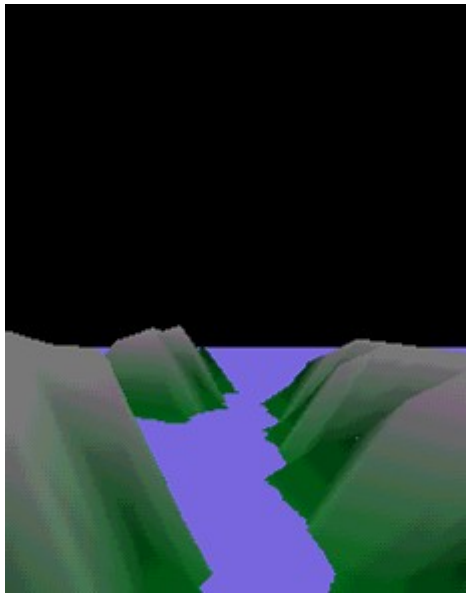
Как всегда, проверьте здесь, если когда-либо Вы растерялись:

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.

Что Вы должны знать

Перед работой с пятой частью обучающей программы Вы должны были прочитать последовательно четыре части, чтобы уже не отставать в написанном ранее коде, который здесь я буду использовать.

3D ландшафт

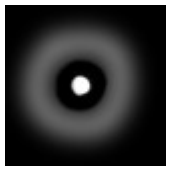


Давайте сначала, определим ландшафт, не так ли? Ландшафт - модель реального мира, с гладкими поверхностями, горами, реками, утесами, холмами … Вы назовете этот ряд. Точка ландшафта должна дать впечатление, что пользователь фактически бродит в "реальном" или "реалистическом" мире. Однако, если мы смотрим на это с более абстрактной точки зрения, мы быстро понимаем, что ландшафт –это только изменения в высоте. Например, травянистая равнина - ландшафт с постоянной высотой (если бы не возможно некоторые впадины и холмы). Область гор – это ландшафт, который имеет очень большие изменения высоты, создавая большие глифы между областями и таким образом создавая иллюзию гор. Река – это равнина, объединенная с тянущейся кривой, которая содержит намного более низкую высоту чем окружающая равнина. Проверьте это изображение ландшафта: Как Вы можете видеть, вышеупомянутый ландшафт описан тремя областями большей высоты (три серых холма), а остальное - глубокое ущелье, которое заполнено водой. Снова, только изменения в высоте.

Карта высот

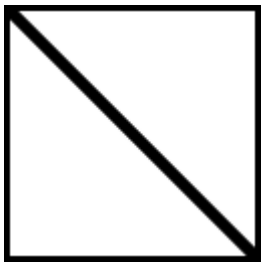
Это - то, где карты высот входят. Они - очень изящные решения хранения изменений высот и для

создания гладких поверхностей. Теперь давайте посмотрим на это изображение прежде, чем я начну что - нибудь показывать.



Это - изображение "градация серого". Ничего из себя не представляет. Это напоминает пончик с белым пятнышком в середине. Так, что является изображением "градация серого? Хорошо проще говоря, это - коллекция пикселов, когда каждый пиксел изменяется от 0 до 255 на шкалу серого, где 0 черный цвет и 255 полностью белый цвет. Правильно? Это звучит знакомым? Что, если Вы могли бы использовать каждый пиксел, чтобы определить высоту тогда? Если черный пиксел (оценивает 0), могла бы быть самая низкая высота и белый пиксел (оценить 255) самый высокий, Вы имели бы карту, которая изображает высоты, карту высот! Другая большая вещь в том, что, так как пикселы изменяются от 0 до 255, Вы получаете автоматическую интерполяцию ландшафта (таким образом создается гладкий ландшафт), если Вы только пятнаете ваше изображение. Так, все, что Вы должны сделать, это открыть вашу любимую графическую программу, начертить кистью некоторый белый материал, и Вы получите карту высот. Это все звучит легко, уверенно, но как мы фактически конвертируем это в Мешь, чтобы мы могли рендерить?

Квадраты



Преобразование карты высот во что-нибудь, что мы можем рендерить, вообще не трудно. Мы должны читать пикселы карты высот и затем создавать поверхности, которые отражают изменения в высоте. Очень удобная в работе поверхность в этой проблеме - конечно квадрат. Так как все изображения прямоугольны, квадраты, действительно хороши для них. Так, что является квадратами? Просто, квадрат - только два треугольника, соединенные, чтобы сформировать прямоугольную поверхность.

Изображение выше представляет квадрат, который состоит из двух треугольников. Как Вы можете видеть, квадрат имеет четыре конечных точки, которые являются необъединенными, так как мы используем два треугольника, чтобы представить его. Этим четырем углам можно дать различные высоты, и таким образом мы имеем начало описания высот в 3D-мире. Одного квадрата недостаточно, чтобы описать полный ландшафт и мы будем нуждаться в их большом количестве, если мы хотим, чтобы наш ландшафт выглядел чуточку реалистичнее. Мы вернемся к этому позже, сначала давайте посмотрим, как мы в коде создаем квадрат. Мы будем делать это x-z планом и с переменными y-координатами, таким образом имея изменяющуюся высоту. Давайте введем новый метод в класс MeshFactory, который мы создавали в предыдущих обучающих программах и назавем его createQuad. Все что этот метод должен знать – это различные высоты в разных углах квадрата, и флаги отбора. Вот - первая часть метода:

```
public static Mesh createQuad(short[] heights, int cullFlags)
   {
      // The vertrices of the quad
      short[] vertrices = {-255, heights[0], -255,
            255, heights[1], -255,
            255, heights[2], 255,
            -255, heights[3], 255};
```

Взгляните, что делается, знакомо? Простой квадрат, состоящий из четырех вершин и каждая имеет изменяющиеся y-компоненты при статических x и z.

```
      // Создать вершины цветов модели
       VertexArray colorArray = new VertexArray(color.length/3, 3, 1);
       colorArray.set(0, color.length / 3, color);

      // Составить VertexBuffer из предыдущего вершин и координат текстур
       VertexBuffer vertexBuffer = new VertexBuffer();
       vertexBuffer.setPositions(vertexArray, 1.0f, null);
       vertexBuffer.setColors(colorArray);

      // Создание индексов и длин граней
      int indices[] = new int[] {0, 1, 3, 2};
       int[] stripLengths = new int[] {4};
```

```
// Создание треугольников моделей
 triangles = new TriangleStripArray(indices, stripLengths);
```

Здесь мы создавали массивы, необходимые для того, чтобы описать Мешь в M3G системе. VertexBuffer содержит два массива вершин, массив цветов и массив позиций. Я преднамеренно не показал массив цветов в вышеупомянутом коде, так как я буду говорить о нем позже. Прямо сейчас сфокусируемся на создании квадрата.

```
// Создание внешности
    Appearance appearance = new Appearance();
    PolygonMode pm = new PolygonMode();
    pm.setCulling(cullFlags);
    pm.setPerspectiveCorrectionEnable(true);
    pm.setShading(PolygonMode.SHADE_SMOOTH);
    appearance.setPolygonMode(pm);
```

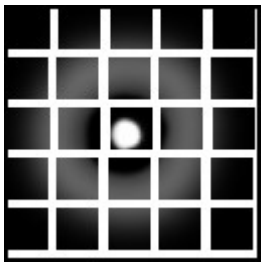
Здесь - некоторый стандартный материал для внешности, однако я хотел бы, чтобы Вы видели, что теперь мы используем гладкую штриховку(SHADE_SMOOTH), что означает - цвета вершин будут интерполированы по целой поверхности, создавая гладкую внешность. Почему мы в этом нуждаемся, станет ясным позже. Все, что осталось, создать Мешь, которая является довольно прямой:

```
// Наконец создайте Мешь
    Mesh mesh = new Mesh(vertexBuffer, triangles, appearance);
```

Создание Квадратов для карты высот

С вышеупомянутым методом мы можем создать Квадрат с изменяющейся высотой, но поскольку я сказал ранее, мы нуждаемся в большом количестве квадратов, чтобы делать реалистический ландшафт то, теперь проблема - как конвертировать карты высот в квадраты. Это - действительно не проблема. Только посмотрите на это изображение:

Я начертил белую сетку по предыдущей карте высот. Если Вы посмотрите на каждую часть сетки, Вы можете увидеть, что прямоугольный сектор сетки - другая меньшая карта высот. Если мы создаем сетку с очень высоким разрешением, Вы вероятно поймете, что сектора сетки становятся очень маленькими и таким образом очень легко приближаются к единичному квадрату. Вывод прост; чтобы аппроксимировать карту высот, мы раскалываем ее на многие очень маленькие части, каждая из которых представляет квадрат. Как мы создаем квадрат? Легко, вот - необходимые шаги:



- Делим изображение на большое число маленьких секторов (минимальный размер 2x2 пиксела)
- Берем пикселы угла каждого сектора и читают их значения (0-255)
- Назначаем эти значения высотами Квадрата (см. декларацию метода),

Так, действительно просто создать квадраты карты высот.
После создания, Вы только рендерите эти квадраты, один за другим и Вы имеете вашу карту высот. Есть некоторые вещи, которые Вы должны знать. Чем больше разрешение сетки карты высот, тем глаже ландшафт, поскольку Вы используете больше квадратов, чтобы представить ландшафт. Однако Вы также сильно увеличиваете расход памяти и увеличиваете число многоугольников, которые GPU должен обработать. Это всегда - компромисс, который должен быть сделан на каждом мобильном телефоне в зависимости от доступной памяти, мощности GPU, и т.д.

Реализация

Давайте посмотрим, как осуществить карту высот в M3G. Мы уже имеем метод, который создает Квадраты с изменяющимися высотами так, что нам необходимо:

1. Загрузить карту высот
2. Создать новый массив, который измерен пропорционально размеру сетки
3. Читать пикселы из карты высот и сохранить в новом массиве
4. Использовать указанный массив, для генерации Квадратов с изменяющимися высотами

Это - простая процедура с четырьмя шагами. Давайте начнем с рассмотрения private членов класса HeightMap:

```
// Фактически карта высот содержит Y-координаты наших треугольников

    private short[] heightMap;
    private int[] data;
    private int imgw, imgh;

    // Размеры карты
    private int mapWidth;
    private int mapHeight;

    // Актуальные квадраты
    private Mesh[][] map;

    // Вода
    private Mesh water;

    // Локальные преобразования, используемые для внутренних вычислений
    private Transform localTransform = new Transform();
```

Они довольно очевидны, но я пройду некоторые из них. Прежде всего массив heightMap - измеренный массив, который содержит высоты. Это не содержит пикселы из изображения карты высот. Таблица Мешей содержит все сгенерированные Квадраты, которые мы рендерим. Наконец, водный Мешь - просто синий план, который представит воду в нашем ландшафте (для того, чтобы создать реки и т.д). Давайте посмотрим, как мы создаем HeightMap:

```
public HeightMap(String imageName, float resolution, int waterLevel) throws IOException
    {
        // Поверка на правильность значения разрешения
        if(resolution <= 0.0001f || resolution > 1.0f)
            throw new IllegalArgumentException("Resolution too small or too large");

        // Загрузка изображения и выделение памяти для внутреннего массива
loadImage(imageName, resolution);

        // Создание квадратов
        createQuads();

        // Создание воды
        createWater(waterLevel);
    }
```

Давайте рассечем вышеупомянутый код на шаги. Сначала мы проверяем на правильность значения разрешения. Недействительные значения ценности - значения больше 1.0f (квадрат имеет 4 угла, таким образом наименьший сектор сетки - 2x2) и значения меньше 0.0001f (ОЧЕНЬ низкое разрешение, которое более или менее создает полный ландшафт с одним Квадратом).
Затем мы загружаем изображение, которое мы передали как параметр конструктора. Однако, есть некоторые другие интересные вещи, сделанные в loadImage методе и я хочу, чтобы Вы их увидели. Вот - код:

```
// Загрузить фактическое изображение
       Image img = Image.createImage(path);

        // Выделим временную память для хранения пикселей
        data = new int[img.getWidth() * img.getHeight()];

        // Получаем их rgb - значения
        img.getRGB(data, 0, img.getWidth(), 0, 0, img.getWidth(), img.getHeight());

        imgw = img.getWidth();
        imgh = img.getHeight();

        // Очищаем память
```

```
      img = null;
      System.gc();

      // Вычисляем новые ширину и высоту
      mapWidth = (int)(res * imgw);
      mapHeight = (int)(res * imgh);

      // Выделяем память для карты высот
      heightMap = new short[mapWidth * mapHeight];

      // Вычисляем высоты и смещение ширины в изображении
      int xoff = imgw / mapWidth;
      int yoff = imgh / mapHeight;

      // Устанавливаем значения высот
      for(int y = 0; y < mapHeight; y++)
      {
         for(int x = 0; x < mapWidth; x++)
         {
            heightMap[x + y * mapWidth] = (short)((data[x * xoff + y * yoff * imgw] & 0x000000ff) *
10);
         }
      }

      // Очищаем память
      data = null;
  img = null;
      System.gc();
```

Я не буду входить в детали в вышеупомянутом коде, поскольку я хотел бы, чтобы Вы рассмотрели его как упражнение. Во всяком случае, мы сначала загружаем фактическое изображение в память и затем извлекаем ее значения пикселов. Затем используя параметр разрешения из конструктора, мы создаем сетку согласующегося размера и заполняем ее значениями пикселов. Наконец, чтобы избавиться от ненужных данных, мы делаем ручную уборку мусора. Это - главным образом потому, что метод loadImage - метод интенсивно использующий память, и мы хотим удостовериться, что данные мусора не уменьшают оперативную память для следующих задач.

Следующий метод в теле конструктора - createQuads. Это - очень прямой метод, который берет сгенерированный массив heightMap(карту высот) и создает из него квадраты. Давайте посмотрим на его внутренности:

```
private void createQuads()
   {
      map = new Mesh[mapWidth][mapHeight];
      short[] heights = new short[4];

      for(int x = 0; x < (mapWidth - 1); x++)
      {
         for(int y = 0; y < (mapHeight - 1); y++)
         {
            // Устанавливаем высоту
            setQuadHeights(heights, x, y, mapWidth);

            // Создаем Мешь
            map[x][y] = MeshFactory.createQuad(heights, PolygonMode.CULL_NONE);
         }
      }
   }
```

Как Вы видите, все, что мы делаем – итеративно по таблице heightMap извлекаем 4 значения, которые мы используем как значения высоты в методе MeshFactory.createQuad.

Я оставлю Вам проверку метода createWater. Это должно быть кое-что, что Вы уже знаете наизусть. Мы только используем MeshFactory.createPlane метод создать большой план, текстурированный текстурой воды.

Рендеринг
Как мы рендерим Квадраты, что мы сгенерировали? Вы должны знать ответ на этот вопрос, но так или иначе давайте пройдем метод render класса HeightMap. Он таков:

```
public void render(Graphics3D g3d, Transform t)
   {
      for(int x = 0; x < map.length - 1; x++)
      {
         for(int y = 0; y < map[x].length - 1; y++)
         {
            localTransform.setIdentity();
            localTransform.postTranslate(x * 5.0f, 0.0f, (mapHeight - y) * -5.0f);
            localTransform.postMultiply(t);
            g3d.render(map[x][y], localTransform);
         }
      }

      localTransform.setIdentity();
      localTransform.postScale(255, 255, 255);
      localTransform.postRotate(-90, 1.0f, 0.0f, 0.0f);
      g3d.render(water, localTransform);
   }
```

Все, что Вы должны сделать – пройти по таблице квадратов и рендерить в их позиции в пространстве. Пользователь render метода может снабдить трансформацией, применительно к каждому квадрату после локальных преобразований, которые только помещают каждый квадрат в его собственное место. Наконец мы размещаем Мешь воды на уровне высоты, определенной во время создания heightmap.

Размещение всего рассмотренного вместе
Теперь, чтобы использовать очень изящный класс HeightMap мы должны сделать следующее:
1. Загрузить HeightMap из существующего изображения “градация серого”
2. Рендерить это
Легко звучит? Давайте посмотрим на код, который загружает HeightMap:

```
private void createScene()
   {
      try
      {
        // Мы используем довольно высокое решение. Если Вы хотите проверить это на фактическом
         // телефоне, пробуйте использовать более низкое решение, типа 0.20 или 0.10

       hm = new HeightMap("/res/heightmap4.png", 0.30f, 40);

       t.postTranslate(0.0f, -2.0f, -5.0f);
       t.postScale(0.01f, 0.01f, 0.01f);

       camTrans.postTranslate(0.0f, 5.0f, 0.0f);
       //camTrans.postTranslate(0.0f, 5.0f, 2.0f);
      }
      catch(Exception e)
      {
         System.out.println("Heightmap error: " + e.getMessage());
         e.printStackTrace();
         TutorialMidlet.die();
      }
   }
```

Здесь ничего странного. Мы только загружаем heightmap и исполняем некоторые трансформации на нем, поскольку эти трансформации поставляются render методу HeightMap's. Мы только хотим вернуть назад в экран бит за битом. Мы также масштабируем его большой размер, поскольку ландшафт обычно огромен, но я хочу, чтобы Вы видели только его маленький краткий обзор.

Другая очень важная вещь, которую имейте в виду - то, что HeightMap в этой обучающей программе рендерит вообще без любого отбора("куллинга"). Это необходимо, особенно на больших ландшафтах. Однако, чтобы держать ясность в коде я хотел удалить любой вид разделения пространства или программного обеспечения отбора. Вы можете видеть это как упражнение, чтобы только послать на рендеринг Меши, которые являются видимыми (то есть не Меши, которые расположены слишком далеко, или позади камеры).
Наконец, это код для рендеринга HeightMap? Вот главный метод draw(черчения):

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

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

// Очистка фона
g3d.clear(back);

// Связать камеру с фиксированной позицией в origo
g3d.setCamera(cam, camTrans);

// Рендерим что-нибудь
hm.render(g3d, t);

// Проверка клавиш управления перемещением камеры
if(key[UP])
{
   camTrans.postTranslate(0.0f, 1.0f, 0.0f);
}
if(key[DOWN])
{
   camTrans.postTranslate(0.0f, -1.0f, 0.0f);
}
if(key[LEFT])
{
   camTrans.postRotate(5, 0.0f, 1.0f, 0.0f);
}
if(key[RIGHT])
{
   camTrans.postRotate(-5, 0.0f, 1.0f, 0.0f);
}

// Полет вперед
if(key[FIRE])
   camTrans.postTranslate(0.0f, 0.0f, -1.0f);
```

Не очень, чтобы сказать реально. HeightMap.render(g3d, t) метод является довольно чистым и прямым. Управление может быть немного нечетко для Вас. Вы перемещаете камеру с джойстиком. Вверх, Вниз и вращаете налево и направо. Чтобы фактически передвигать камеру вперед, используйте клавишу FIRE.

Заключение
Так, для продолжения этого урока, почему бы Вам не попробовать загрузить другой heightmaps, вложенный в Zip-файл с исходным кодом? Смотрите какие ландшафты получаются. Или еще лучше; создайте ваше собственное изображение heightmap!
Позвольте вашему воображению понервничать, поместив его в MIDlet и сделав круиз через ваш пейзаж.

TutorialMidlet

```
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 TutorialMidlet extends MIDlet implements CommandListener
{
    // Переменная, которая держит уникальный display
 private Display display = null;

 // Холст
 private M3GCanvas canvas = null;

 // Ссылка в MIDlet-е на себя самого
 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.Background;
import javax.microedition.m3g.Camera;
import javax.microedition.m3g.Graphics3D;
import javax.microedition.m3g.Transform;

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.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[9];

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

    // Глобальный объект Graphics3D
    Graphics3D g3d = null;

   // Фон
    Background back = null;

    // Глобальный объект камеры
    Camera cam = null;
       // Трансформации
 Transform camTrans = new Transform();



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

       // Мы желаем полноэкранный холст
       setFullScreenMode(true);

       // Создание сцены
       createScene();
```

```
    // Загрузка нашей камеры
    loadCamera();

    // Загрузка фона
    loadBackground();

        // Устанавливаем 3D-графику
    setUp();
}

/ ** Готовит движок Graphics3D к непосредственному режиму рендеринга, добавляя свет */
   private void setUp()
{
    // Получить инстанцию
    g3d = Graphics3D.getInstance();
}
        / ** 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 нашей камеры */
private void loadCamera()
{
    // Создание новой камеры
    cam = new Camera();

    // Установить перспективу нашей камеры. Мы управляем установкой дальнего плана
    // (текущее 150.0f) и с наименьшим значением 50.0f
    cam.setPerspective(60.0f, (float)getWidth() / (float)getHeight(), 0.1f, 150.0f);
}

/** Загрузка фона */
private void loadBackground()
{
    // Создание нового фона, и установка его цвета в черный цвет
    back = new Background();
    back.setColor(0);
}

// Карта высот и ее трансформация
HeightMap hm;
Transform t = new Transform();

/** Создание нашей сцены */
private void createScene()
{
    try
    {
      // Мы используем довольно высокое решение. Если Вы хотите проверить это на фактическом
       // телефоне, пробуйте использовать более низкое решение, типа 0.20 или 0.10
     hm = new HeightMap("/res/heightmap4.png", 0.30f, 40);

     t.postTranslate(0.0f, -2.0f, -5.0f);
     t.postScale(0.01f, 0.01f, 0.01f);
```

```
      camTrans.postTranslate(0.0f, 5.0f, 0.0f);
      //camTrans.postTranslate(0.0f, 5.0f, 2.0f);
     }
     catch(Exception e)
     {
        System.out.println("Heightmap error: " + e.getMessage());
        e.printStackTrace();
        TutorialMidlet.die();
     }
  }

/** Чертим на экране
   */
    private void draw(Graphics g)
  {
    // Окутать все пробующим/ловящим блоком на всякий случай
     try
     {
       // Получить Graphics3D контекст
        g3d = Graphics3D.getInstance();

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

      // Очистка фона
      g3d.clear(back);

      // Связать камеру с фиксированной позицией в origo
      g3d.setCamera(cam, camTrans);

     // Рендерим что-нибудь
      hm.render(g3d, t);

      // Проверка клавиш управления перемещением камеры
            if(key[UP])
      {
         camTrans.postTranslate(0.0f, 1.0f, 0.0f);
      }
      if(key[DOWN])
      {
         camTrans.postTranslate(0.0f, -1.0f, 0.0f);
      }
      if(key[LEFT])
      {
         camTrans.postRotate(5, 0.0f, 1.0f, 0.0f);
      }
      if(key[RIGHT])
      {
         camTrans.postRotate(-5, 0.0f, 1.0f, 0.0f);
      }

      // Полет вперед
      if(key[FIRE])
         camTrans.postTranslate(0.0f, 0.0f, -1.0f);
     }
     catch(Exception e)
     {
        reportException(e);
     }
     finally
     {
        // Всегда помните отвязать!
        g3d.releaseTarget();
```

```
    }
}
  /** 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; }
}
```

MeshFactory

```
import java.io.IOException;

import javax.microedition.lcdui.Image;
import javax.microedition.m3g.Appearance;
import javax.microedition.m3g.Image2D;
import javax.microedition.m3g.IndexBuffer;
import javax.microedition.m3g.Light;
import javax.microedition.m3g.Material;
import javax.microedition.m3g.Mesh;
import javax.microedition.m3g.PolygonMode;
import javax.microedition.m3g.Texture2D;
import javax.microedition.m3g.TriangleStripArray;
import javax.microedition.m3g.VertexArray;
import javax.microedition.m3g.VertexBuffer;

/**
 * Статический класс, который управляет созданием сгенерированных кодом Мешей
 */
public class MeshFactory
{
    /** Создает план текстуры, который альфа-смешан
     *
      * @param texFilename Имя файла изображения текстуры
     * @param cullFlags Флаги для куллинга. Смотри PolygonMode.
     * @param alpha Альфа –значение для смешивания. Формат полного цвета - 0xAARRGGBB
     * @return Законченный текстурированный мешь

     */
    public static Mesh createAlphaPlane(String texFilename, int cullFlags, int alpha)
    {
```

```
    // Создать нормали меша
    Mesh mesh = createPlane(texFilename, cullFlags);

    // Сделать его смешивающимся
    MeshOperator.convertToBlended(mesh, alpha, Texture2D.FUNC_BLEND);

    return mesh;
}

/**
 * Создает цветной квадрат, состоящий из двух треугольников.
 * Параметр heights – массив содержащий высоты нового квадрата.
 * Массив должен иметь длину 4 и начинаться против часовой стрелки из
 * северо-западной позиции квадрата (-1, -1)
 * @param heights – массив высот, замечание высот(y-координаты) четырех углов.
 * @param cullFlags
 * @param texFilename
 * @return
 */
public static Mesh createQuad(short[] heights, int cullFlags)
{
    // Вершины квадрата
    short[] vertrices = {-255, heights[0], -255,
            255, heights[1], -255,
            255, heights[2], 255,
            -255, heights[3], 255};

    // Массивы
    VertexArray vertexArray;
    IndexBuffer triangles;

    // Создание вершин модели
    vertexArray = new VertexArray(vertrices.length/3, 3, 2);
    vertexArray.set(0, vertrices.length/3, vertrices);

    // Выделение массива цветов
    byte[] color = new byte[12];

    for(int i = 0; i < heights.length; i++)
    {
       int j = i * 3;
     // Проверка высот
       if(heights[i] >= 1000)
       {
          byte col = (byte)(57 + (heights[i] / 1550.0f) * 70);
          color[j] = col;
          color[j + 1] = col;
          color[j + 2] = col;
       }
     else
     {
        byte gCol = 110;
        byte bCol = 25;

        color[j] = 0;
        color[j + 1] = (byte)(gCol - (heights[i] / 1000.0f) * 85);
        color[j + 2] = (byte)(bCol - (heights[i] / 1000.0f) * 20);
     }
    }

    // Создание вершин цвета модели
    VertexArray colorArray = new VertexArray(color.length/3, 3, 1);
    colorArray.set(0, color.length / 3, color);
```

```
    // Смешивание предыдущих вершин  из VertexBuffer и координат текстуры
    VertexBuffer vertexBuffer = new VertexBuffer();
    vertexBuffer.setPositions(vertexArray, 1.0f, null);
    vertexBuffer.setColors(colorArray);

    // Создание индексов и длины полосы
    int indices[] = new int[] {0, 1, 3, 2};
    int[] stripLengths = new int[] {4};

    // Создание полосы треугольников модели
    triangles = new TriangleStripArray(indices, stripLengths);

    // Создание Внешности
    Appearance appearance = new Appearance();
    PolygonMode pm = new PolygonMode();
    pm.setCulling(cullFlags);
    pm.setPerspectiveCorrectionEnable(true);
    pm.setShading(PolygonMode.SHADE_SMOOTH);
    appearance.setPolygonMode(pm);

    // Наконец создаем Мешь
    Mesh mesh = new Mesh(vertexBuffer, triangles, appearance);

    // All done
    return mesh;

}

/**
 * @param texFilename
 * @return
 * @throws IOException
 */
public static Texture2D createTexture2D(String texFilename) throws IOException {
    // Открыть изображение
    Image texImage = Image.createImage(texFilename);
    Texture2D theTexture = new Texture2D(new Image2D(Image2D.RGBA, texImage));

    // Установка смешивания -Modulate
    theTexture.setBlending(Texture2D.FUNC_MODULATE);

    // Установка обертывания и фильтрации
    theTexture.setWrapping(Texture2D.WRAP_CLAMP, Texture2D.WRAP_CLAMP);
    theTexture.setFiltering(Texture2D.FILTER_NEAREST, Texture2D.FILTER_NEAREST);
    return theTexture;
}

/**
 * Создание плана текстуры.
 * @param texFilename Имя файла образа текстуры
 * @param cullFlags Флаги для куллинга. Смотрите PolygonMode.
 * @return Окончательный текстурированный Мешь
 */
public static Mesh createPlane(String texFilename, int cullFlags)
{
    // The vertrices of the plane
    short vertrices[] = new short[] {-1, -1, 0,
                                     1, -1, 0,
                                     1, 1, 0,
                                     -1, 1, 0};

    // Координаты плана текстуры
    short texCoords[] = new short[] {0, 255,
```

```
                                255, 255,
                                255, 0,
                                0, 0};


        // Классы
        VertexArray vertexArray, texArray;
        IndexBuffer triangles;

        // Создание вершин модели
        vertexArray = new VertexArray(vertrices.length/3, 3, 2);
        vertexArray.set(0, vertrices.length/3, vertrices);

        // Создание координат текстуры модели
        texArray = new VertexArray(texCoords.length / 2, 2, 2);
        texArray.set(0, texCoords.length / 2, texCoords);

        // Смешивание предыдущих вершин  из VertexBuffer и координат текстуры
        VertexBuffer vertexBuffer = new VertexBuffer();
        vertexBuffer.setPositions(vertexArray, 1.0f, null);
        vertexBuffer.setTexCoords(0, texArray, 1.0f/255.0f, null);

       // Создание индексов и длины полосы
        int indices[] = new int[] {0, 1, 3, 2};
        int[] stripLengths = new int[] {4};

        // Создание полосы треугольников модели
        triangles = new TriangleStripArray(indices, stripLengths);

        // Создание Внешности
        Appearance appearance = new Appearance();
        PolygonMode pm = new PolygonMode();
        pm.setCulling(cullFlags);
        appearance.setPolygonMode(pm);

        // Создание и установка текстуры
        try
        {
           // Открытие изображения
           Texture2D theTexture = createTexture2D(texFilename);

           // Добавление текстуры к внешности
           appearance.setTexture(0, theTexture);

        }
        catch(Exception e)
        {
           // Кое-что пошло не так, как надо
           System.out.println("Failed to create texture");
           System.out.println(e);
        }

        // Наконец создаем Мешь
        Mesh mesh = new Mesh(vertexBuffer, triangles, appearance);

        // All done
        return mesh;
    }
}
```

MeshOperator

```
import javax.microedition.m3g.CompositingMode;
import javax.microedition.m3g.Mesh;
/**
 * Исполняет некоторые основные действия на Мешь-объектах
 */
public class MeshOperator
{
   /** Устанавливает альфа-смешивание Мешу. Только имеет значение,

    * если Мешь уже – альфа смешивающийся

*/
   public static void setMeshAlpha(Mesh m, int alpha)
   {
      m.getVertexBuffer().setDefaultColor(alpha);
   }

   /**
    *
    * @param m Мешь, конвертируемый в смешивающийся
    * @param alpha Альфа цвет для смешивания
    * @param textureBlending Параметр смешивания текстуры.
    */
   public static void convertToBlended(Mesh m, int alpha, int textureBlending)
   {
      // Установка альфы
      setMeshAlpha(m, alpha);

      // Установка режима смешивания
      CompositingMode cm = new CompositingMode();
      cm.setBlending(CompositingMode.ALPHA);
      m.getAppearance(0).setCompositingMode(cm);
      m.getAppearance(0).getTexture(0).setBlending(textureBlending);
   }

   public static void setPerspectiveCorrection(Mesh m, boolean on)
   {
      m.getAppearance(0).getPolygonMode().setPerspectiveCorrectionEnable(on);
   }
}
```

HeightMap

```
import java.io.IOException;

import javax.microedition.lcdui.Image;
import javax.microedition.m3g.Graphics3D;
import javax.microedition.m3g.Mesh;
import javax.microedition.m3g.PolygonMode;
import javax.microedition.m3g.Transform;

/**
 *
 */
public class HeightMap
{
   // Актуальная heightmap, содержащая Y-координаты нашихтреугольников
   private short[] heightMap;
   private int[] data;
   private int imgw, imgh;

   // Размеры карты
```

```
private int mapWidth;
private int mapHeight;

// Актуальные квадраты
private Mesh[][] map;

// Вода
private Mesh water;

// Локальная трансформация, используемая для внутренних вычислений
private Transform localTransform = new Transform();

/**
 * Выделение внутренней heightmap и подготовка ее для рендеринга.
 * @param imageName Путь к  файлу черно-белого изображения heightmap
 * @param resolution Разрешение heightmap. Значение 1.0 -  каждый пиксел - один квадрат.
 * @param waterLevel TODO
 * @throws IOException если ошибка загрузки изображений heightmap или текстуры.
 */
public HeightMap(String imageName, float resolution, int waterLevel) throws IOException
{
   // Проверка на правильность  Разрешения
   if(resolution <= 0.0001f || resolution > 1.0f)
      throw new IllegalArgumentException("Resolution too small or too large");

   // Загрузка изображения и выделение памяти для внутреннего массива
   loadImage(imageName, resolution);

   // Создание квадратов
   createQuads();

   // Создание воды
   createWater(waterLevel);
}

/** Создание воды  в очень простой манере, представляя ее
 * текстурированным квадратом.
 */
private void createWater(int level) throws IOException
{
   water = MeshFactory.createPlane("/res/water0.png", PolygonMode.CULL_NONE);
}

/**
 * Создание всех квадратов(Части их - треугольники)
 */
private void createQuads()
{
   map = new Mesh[mapWidth][mapHeight];
   short[] heights = new short[4];

   for(int x = 0; x < (mapWidth - 1); x++)
   {
      for(int y = 0; y < (mapHeight - 1); y++)
      {
         // Установка высот
         setQuadHeights(heights, x, y, mapWidth);

         // Создание меша
         map[x][y] = MeshFactory.createQuad(heights, PolygonMode.CULL_NONE);
      }
   }
}
```

```
  /**
   * @param высоты
   * @param x
   * @param y
   */
  private void setQuadHeights(short[] heights, int x, int y, int scanline)
  {
     heights[0] = heightMap[x + y * scanline];
     heights[1] = heightMap[x + y * scanline + 1];
     heights[3] = heightMap[x + (y + 1) * scanline];
     heights[2] = heightMap[x + (y + 1) * scanline + 1];
  }

  private void loadImage(String path, float res) throws IOException
  {
     // Загрузка актуального изображения
     Image img = Image.createImage(path);

     // Выделение временной памяти для запоминания пикселей
     data = new int[img.getWidth() * img.getHeight()];

     // Получение их rgb - значений
     img.getRGB(data, 0, img.getWidth(), 0, 0, img.getWidth(), img.getHeight());

     imgw = img.getWidth();
     imgh = img.getHeight();

     // Очистка памяти
     img = null;
     System.gc();

     // Вычисление новых ширины и высоты
     mapWidth = (int)(res * imgw);
     mapHeight = (int)(res * imgh);

     // Выделение памяти под heightmap
     heightMap = new short[mapWidth * mapHeight];

     // Вычисление ширины и высоты смещения в изображении

     int xoff = imgw / mapWidth;
     int yoff = imgh / mapHeight;

     // Установка значений высоты
     for(int y = 0; y < mapHeight; y++)
     {
        for(int x = 0; x < mapWidth; x++)
        {
           heightMap[x + y * mapWidth] = (short)((data[x * xoff + y * yoff * imgw] & 0x000000ff) *
10);
        }
     }

     // Освобождение памяти
     data = null;
     img = null;
     System.gc();
  }

  /**
   * Рендеринг этой heightmap, используя специальный графический контекст и трансформацию.
   * @param g3d
   * @param t
   */
```

```
    public void render(Graphics3D g3d, Transform t)
    {
        for(int x = 0; x < map.length - 1; x++)
        {
            for(int y = 0; y < map[x].length - 1; y++)
            {
                localTransform.setIdentity();
                localTransform.postTranslate(x * 5.0f, 0.0f, (mapHeight - y) * -5.0f);
                localTransform.postMultiply(t);
                g3d.render(map[x][y], localTransform);
            }
        }

        localTransform.setIdentity();
        localTransform.postScale(255, 255, 255);
        localTransform.postRotate(-90, 1.0f, 0.0f, 0.0f);
        g3d.render(water, localTransform);
    }
}
```

О Redikod

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. Более подробно на WEB-сайте Redikod>>.

P.S.