Using M3G to draw in Java ME

Overview

This code sample shows how to use Mobile 3D Graphics API for simple drawing. It represents using simple geometry, material, light, background, rendering, and camera settings.

For drawing possibility we must create our implementation of `Canvas` object (Image object can also be used). It consist of initialisation (`initialize()` method with initialisation of scene objects) and rendering (`paint()` method) functionality.

Source file: SimpleDrawingUsingM3G.java

```
import java.io.IOException;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

import javax.microedition.m3g.Appearance;
import javax.microedition.m3g.Background;
import javax.microedition.m3g.Camera;
import javax.microedition.m3g.Graphics3D;
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.Texture2D;
import javax.microedition.m3g.Transform;
import javax.microedition.m3g.TriangleStripArray;
import javax.microedition.m3g.VertexArray;
import javax.microedition.m3g.VertexBuffer;
import javax.microedition.midlet.MIDlet;


/**
 *
 */
public class DrawingMIDlet extends MIDlet {
    /**
     * Reference to MIDlet instance for using it inside Canvas3D
     */
    static DrawingMIDlet instance;
    /**
     * Canvas instance using for drawing with m3g
     */
    Canvas3D canvas3d;/**
     * Boolean variable to prevent paint() method to be called more than once
     */
    private boolean isPainted;
    /**
     * The Canvas3D constructor.
     */
    public DrawingMIDlet() {
        // Set up the user interface.
        DrawingMIDlet.instance = this;
        canvas3d = new Canvas3D();
        isPainted = true;
    }

    /**
     * Returns a display instance.
     * @return the display instance.
     */
    public Display getDisplay () {
        return Display.getDisplay(this);
    }
```


(C) Copyright Nokia 2013. All rights reserved.

```
    /**
     * From MIDlet.
     * Called when MIDlet is started.
     * Sets object from canvas3d variable as current displayable.
     */
    public void startApp() {
        Display.getDisplay(this).setCurrent(canvas3d);
        if( canvas3d != null ) {
            canvas3d.repaint();
        }
    }
    /**
     * From MIDlet.
     * Called when MIDlet is paused.
     */
    public void pauseApp() {
        //Empty implementation
    }

    /**
     * From MIDlet.
     * Called to signal the MIDlet to terminate.
     * @param unconditional if true, then the MIDlet has to be unconditionally
     * terminated and all resources has to be released.
     */
    public void destroyApp(boolean unconditional) {
        //Empty implementation
    }

    /**
     * Quit the MIDlet.
     */
    public static void quitApp() {
        instance.destroyApp(true);
        instance.notifyDestroyed();
        instance = null;
    }
    /**
     * Inner class for handling the canvas.
     */
    class Canvas3D extends Canvas implements CommandListener{
        /**
         * exit midlet command
        */
        private Command exitCommand;

        /**
         * Reference to 3D graphics context that can be bound to a rendering
         * target. All rendering is done through the render methods
         * in this class
         */
        private Graphics3D iG3D;
        /**
         * Defines the position of the viewer in the scene and
         * the projection from 3D to 2D.
         */
        private Camera iCamera;
        /**
         * Represents different kinds of light sources.
         */
        private Light iLight;
        /**
         * A generic 4x4 floating point matrix, representing a transformation.
         */
        private Transform iTransform;
        /**
         * Defines whether and how to clear the viewport. It can be
         * image or color for solid filling.
         */
        private Background iBackground;
        /**
         * Mesh object that represents a cube
         */
        Mesh iBoxMesh;

        /**
         * Construct the Displayable.
         */
        public Canvas3D() {
            exitCommand = new Command("Exit", Command.EXIT, 1);

            addCommand(exitCommand);

            // set up this Displayable to listen to command events
            setCommandListener(this);

            initialize();
        }

        /**
         * Initialize self.
         */
        void initialize() {
            // get the singleton Graphics3D instance
            iG3D = Graphics3D.getInstance();
            // create a camera object
            iTransform = new Transform();

            iBackground = backgroundInitialization("/background.png");
            //Material is an Appearance component encapsulating material
            //attributes for lighting computations.
            Material material = materialInitialization(0xFFFFFFFF,
                    0xFFFFFFFF, 100.0f);

            iCamera = cameraInitialization(70.0f);
            iLight = lightInitialization(0xFFFFFFFF, Light.SPOT);

            try {
                //creating of cube object as Mesh
                iBoxMesh = cubeMeshInitialization(material);
            }catch(Exception e) {
                //TODO: write handler code
            }

            }

        /**
  * From Canvas.
         * Rendering the scene
         */
```


(C) Copyright Nokia 2013. All rights reserved.

```
        protected void paint(Graphics g) {
            if (isPainted) {
         System.out.println("Called paint method");
         isPainted = false;
         // Bind the Graphics of this Canvas to Graphics3D.
         //We enable depth buffer, dithering and  true color rendering(if it
         //supported of course)
         iG3D.bindTarget(g, true, Graphics3D.DITHER | Graphics3D.TRUE_COLOR);
         // clear the color and depth buffers
         iG3D.clear(iBackground);
         // set up the camera in the desired position
         Transform transform = new Transform();
         transform.postTranslate(0.0f, 0.0f, 35.0f);
         iG3D.setCamera(iCamera, transform);
         // set up a "headlight": a directional light shining
         // from the direction of the camera
         iG3D.resetLights();
         iG3D.addLight(iLight, transform);
         // update our transform (this will give us a rotating cube)

         iTransform.postRotate(28.0f, 0.1f, 1.0f, 0.0f);
         //Renders the cube with the given transformation from local
         //coordinates to world coordinates
         iG3D.render(iBoxMesh, iTransform);

         iG3D.releaseTarget();
         }
        }

        /**
         * When the screen is shut down, the isPainted will be true.
         */
        protected void hideNotify() {
            if (isPainted == false) {
                isPainted = true;
            }
        }

        /**
         * Creating a Material object
         * @param diffuseColor is specifying that the diffuse color component
         * is to be set to a Material object
         * @param specularColor is specifying that the specular color component
         * is to be set to a Material object
         * @param shininess is the specular exponent term in the
         * lighting equation
         * @return the prepeared Material object
         */
        private Material materialInitialization(int diffuseColor,
                int specularColor, float shininess) {
            Material material = new Material();
            try{
                //set color for Material object. if
                material.setColor(Material.DIFFUSE, diffuseColor);
                material.setColor(Material.SPECULAR, specularColor);
                material.setShininess(shininess);
            }catch(Exception e) {
                //TODO: write handler code
            }
            return material;
        }
        /**
         * Method creates a new Background object with image instead of
         * background color
         * @param imageFileName is the name of image file
         * @return the prepeared Background object
         */
        private Background backgroundInitialization(String imageFileName) {
            Background background = new Background();
            try {
                // loading image for background with the same approach as
                //texture image
                Image backImage = null;
                // load the image for the texture
                backImage = Image.createImage(imageFileName);
                Image2D backgroundImage2D =
                        new Image2D( Image2D.RGB, backImage);
                background.setImageMode(Background.BORDER, Background.BORDER);
                background.setImage(backgroundImage2D);
            }catch(Exception e) {
                //TODO: write handler code
            }
            return background;
        }
        /**
         * Creates a camera object with perspective projection
         * @param angleOfView is the camera field of view
         * @return the prepeared Camera object
         */
        private Camera cameraInitialization(float angleOfView) {
            Camera camera = new Camera();
            try{
                camera.setPerspective( angleOfView,
                    (float)getWidth()/ (float)getHeight(), // aspectRatio
                    1.0f, // near clipping plane
                    1000.0f ); // far clipping plane
            }catch(Exception e) {
                //TODO: write handler code
            }
            return camera;
        }
        /**
         * Creates a light source object
         * @param color is the color component of light
         * @param lightType is the type of light source(like SPOT, OMNI and
         * others)
         * @return the prepeared Light object
         */
        private Light lightInitialization(int color, int lightType) {
            // create a light object
            Light light = new Light();
            try{
                // white light. Later this parameter will be animated
                light.setColor(color);
                //Sets the intensity of light
                light.setIntensity(1.0f);
                //sets the spot type for this light
                light.setMode(lightType);
                //sets the spot exponent, that controls the distribution of the
                //intensity of light within the spot cone
```


(C) Copyright Nokia 2013. All rights reserved.

```
            light.setSpotExponent(1.0f);
            //sets the spot angle
            light.setSpotAngle(45);
        }catch(Exception e) {
            //TODO: write handler code
        }
        return light;
    }

    /**
     * Creates Cube Mesh
     * @param material is the Appearance component of new Mesh
     * @return the prepeared Mesh object
     */
    private Mesh cubeMeshInitialization(Material material) {
        // initialize some arrays for our object (cube)
        // Each line in this array declaration represents a triangle
        // strip for one side of a cube.
        // 1 * * * * * 0
        //   * *     *
        //   *   *   *
        //   *     * *
        // 3 * * * * * 2
        // The ascii diagram above represents the vertices in the first line
        // (the first tri-strip)
        short[] vert = {
            10, 10, 10, -10, 10, 10, 10,-10, 10, -10,-10, 10, // front
            -10, 10,-10, 10, 10,-10, -10,-10,-10, 10,-10,-10, // back
            -10, 10, 10, -10, 10,-10, -10,-10, 10, -10,-10,-10, // left
            10, 10,-10, 10, 10, 10, 10,-10,-10, 10,-10, 10, // right
            10, 10,-10, -10, 10,-10, 10, 10, 10, -10, 10, 10, // top
            10,-10, 10, -10,-10, 10, 10,-10,-10, -10,-10,-10 }; // bottom
        // create a VertexArray to hold the vertices for the object
        VertexArray vertArray = new VertexArray(vert.length / 3, 3, 2);
        vertArray.set(0, vert.length/3, vert);
        // normal vectors for the cube. used for lighting
        byte[] norm = {
            0, 0, 127,  0, 0, 127,  0, 0, 127,  0, 0, 127,
            0, 0,-127,  0, 0,-127,  0, 0,-127,  0, 0,-127,
            -127, 0, 0, -127, 0, 0, -127, 0, 0, -127, 0, 0,
            127, 0, 0,  127, 0, 0,  127, 0, 0,  127, 0, 0, 0,
            127, 0, 0,  127, 0, 0,  127, 0, 0,  127, 0, 0,-127,
            0, 0,-127, 0, 0,-127, 0, 0,-127, 0 };

        // create a vertex array for the normals of the object
        VertexArray normArray = new VertexArray(norm.length / 3, 3, 1);
        normArray.set(0, norm.length/3, norm);
        // the length of each triangle strip
        int[] stripLen = { 4, 4, 4, 4, 4, 4 };

        //Creating of instance of TriangleStripArray
        //(derived from IndexBuffer). It
        //represents an array of triangle strips.
        //In a triangle strip, the first three vertex indices define
        //the first triangle. Each subsequent index together with the two
        //previous indices define a new triangle.
        // create the index buffer for our object (this tells how to
        // create triangle strips from the contents of the vertex buffer).
        IndexBuffer indexBuffer = new TriangleStripArray( 0, stripLen );

        // per vertex texture coordinates
        //values represents mapping of texture to vertices of 3d object
        short[] tex = {
            1, 0, 0, 0, 1, 1, 0, 1,
            1, 0, 0, 0, 1, 1, 0, 1,
            1, 0, 0, 0, 1, 1, 0, 1,
            1, 0, 0, 0, 1, 1, 0, 1,
            1, 0, 0, 0, 1, 1, 0, 1,
            1, 0, 0, 0, 1, 1, 0, 1 };
        // create a vertex array for the texture coordinates of the object
        VertexArray texArray = new VertexArray(tex.length / 2, 2, 2);
        texArray.set(0, tex.length/2, tex);
        //VertexBuffer holds references to VertexArrays that contain the
        //positions, colors, normals, and texture coordinates for a set
        //of vertices create the VertexBuffer for our object
        VertexBuffer vertexBuffer = new VertexBuffer();
        vertexBuffer.setPositions(vertArray, 1.0f, null);
        // unit scale, zero bias
        vertexBuffer.setNormals(normArray);
        vertexBuffer.setTexCoords(0, texArray, 1.0f, null);
        //Texture creating
        Image textureImage = null;
        try {
            // load the image for the texture
            textureImage = Image.createImage("/texture.png");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        // create the Image2D (we need this so we can make a Texture2D)
        Image2D image2D = new Image2D( Image2D.RGB, textureImage );
        // create the Texture2D and enable mipmapping
        Texture2D texture = new Texture2D( image2D );
        texture.setFiltering(Texture2D.FILTER_LINEAR,
                             Texture2D.FILTER_LINEAR);
        //specifying that the texture image is to be repeated only once
        texture.setWrapping(Texture2D.WRAP_CLAMP, Texture2D.WRAP_CLAMP);
        // texture color is to be modulated with the lit material color
        texture.setBlending(Texture2D.FUNC_MODULATE);

        // create the appearance object
        //It's a set of component objects that define the rendering
        //attributes. Using during render operation.
        Appearance cubeAppearance = new Appearance();
        cubeAppearance.setTexture(0, texture);
        cubeAppearance.setMaterial(material);

        //creating Mesh object and passing as parameters
        //vertex, index buffers and Appearance component
        Mesh mesh = new Mesh(vertexBuffer, indexBuffer, cubeAppearance);
        return mesh;
    }
/**
 * From CommandListener.
 */
    public void commandAction(Command command, Displayable displayable) {
        if (command == exitCommand) {
            // exit the MIDlet
            DrawingMIDlet.quitApp();
        }
    }
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }
}
```

Postconditions

This MIDlet draws a rotated and textured cube with a background picture.

Supplementary material

You can download this MIDlet, source code, and resources from here: Simple Drawing.zip

See also: Using M3G Animation in Java ME

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.