Using M3G Animation in Java ME

Overview

This code example shows how to use Mobile 3D Graphics API for animation with `KeyframeSequence`, `AnimationTrack`, and `AnimationController` classes.

First, define what kind of animation to use (see `animationInitialization(Light light, Mesh mesh)` method). Then create a `KeyframeSequence` object which consist of some keyframes with appropriate values, and create AnimationTrack object with animation type defining.

Then create a `AnimationController` object, tune it, and add `AnimationTrack` objects to the 3D entity (like Light, Mesh, and others).

Later in `paint()` method the animate function is called for each 3D entity and the scene is rendered.

Animation is added to an already existing snippet Using M3G to draw in Java ME. So here only those new declarations, handlers, methods, and changes are shown which should be performed over a simple drawing snippet.

Source file: AnimationMIDlet.java

```
import javax.microedition.m3g.AnimationController;
import javax.microedition.m3g.AnimationTrack;
import javax.microedition.m3g.KeyframeSequence;

...

/**
 *
 */
public class AnimationMIDlet extends MIDlet {
    /**
     * Reference to MIDlet instance for using it inside Canvas3D
     */
    static AnimationMIDlet instance;

    /**
     * The Canvas3D constructor.
     */
    public AnimationMIDlet() {
      AnimationMIDlet.instance = this;
    }

    ...

    /**
     * From MIDlet.
     * Called when MIDlet is started.
     * Sets object from canvas3d variable as current displayable.
     * Schedules a MyTimerTask timer task.
     */
    public void startApp() {
      Display.getDisplay(this).setCurrent(new Canvas3D());
    }

    ...


    /**
     * Inner class for handling the canvas.
     */
    class Canvas3D extends Canvas implements Runnable,CommandListener{

        ...

        /**
         * absolute counter of animation in world units
         */
        int iApplicationTime;
```


(C) Copyright Nokia 2013. All rights reserved.

```
        /**
         * Mesh object that represents a cube
         */
        Mesh iAnimatedMesh;

        public Canvas3D() {
           ...
           initialize();

           Thread thread = new Thread(this);
           thread.setPriority(Thread.MAX_PRIORITY);
           thread.start();
        }

        /**
         * Initialize self.
         */
        void initialize() {
            // get the singleton Graphics3D instance
            iG3D = Graphics3D.getInstance();

            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
                iAnimatedMesh = cubeMeshInitialization(material);

                animationInitialization(iLight, iAnimatedMesh);
            }catch(Exception e) {
                //TODO: write handler code
            }

            iApplicationTime = 0;
            }

            public void run(){

                int dt =0;
                long loopStartTime =System.currentTimeMillis();

                running = true;

                while(running){
                    dt = (int)(System.currentTimeMillis()-loopStartTime);
                    loopStartTime = System.currentTimeMillis();
                    //incrementing world time
                    iApplicationTime = iApplicationTime + dt;

                    repaint();
                    serviceRepaints();

                    if(System.currentTimeMillis()-loopStartTime<20){
                        try {
                             Thread.sleep(20 - (System.currentTimeMillis() - loopStartTime));
                        } catch (Exception e) {
                             // TODO: handle exception
                        }
                     }
                 }

                 AnimationMIDlet.quitApp();

             }

        /**
    * From Canvas.
         * Rendering the scene
         */
        protected void paint(Graphics g) {
            //incrementing world time
            iApplicationTime = iApplicationTime + 50;

            // 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);

            //calcuulating parameters for animation according to
            //current world time
            iLight.animate(iApplicationTime);

            // update Light object
            iG3D.resetLights();
            iG3D.addLight(iLight, transform);

            transform.setIdentity();
            //calcuulating parameters for animation according to
            //current world time
            iAnimatedMesh.animate(iApplicationTime);
            // In immediate mode, node transforms are ignored, so we get
            // our animated transformation into a local transform object
            iAnimatedMesh.getCompositeTransform(transform);
            // update our transform (this will give us a rotating cube)
            iG3D.render(iAnimatedMesh, transform);

            iG3D.releaseTarget();
        }
        /**
         * From CommandListener.
         * @param command
         * @param displayable
         */
        public void commandAction(Command command, Displayable displayable) {
            if (command == exitCommand) {
                // exit the MIDlet
               running = false;
            }
        }
```


(C) Copyright Nokia 2013. All rights reserved.

```
        ...

        /**
         *
         * @param light is the Light object for color animation
         * @param mesh is the Mesh object for rotation animation
         */
        private void animationInitialization(Light light, Mesh mesh) {
            //Sequence of keyframes for rotation of cube
            //creating color KeyframeSequence object
            //with 5 keyframes with 3 components in each
            //and setting the intepolation type
            KeyframeSequence colorKeyframes = new KeyframeSequence(5, 3,
                    KeyframeSequence.LINEAR);
            //then setting the duration of color animation
            colorKeyframes.setDuration(4000);
            //this animation will play only once without repeat
            colorKeyframes.setRepeatMode(KeyframeSequence.CONSTANT);
            //setting index, timestamp, and components values for 5 keyframes
            colorKeyframes.setKeyframe(0,   0, new float[]{1.0f, 1.0f, 1.0f});
            colorKeyframes.setKeyframe(1, 1000, new float[]{1.0f, 0.0f, 0.0f});
            colorKeyframes.setKeyframe(2, 2000, new float[]{0.0f, 1.0f, 0.0f});
            colorKeyframes.setKeyframe(3, 3000, new float[]{0.0f, 0.0f, 1.0f});
            colorKeyframes.setKeyframe(4, 4000, new float[]{1.0f, 1.0f, 1.0f});

            //Associates a KeyframeSequence with an AnimationController and sets
            //the type of animation. In this case it's color animation.
            //creating AnimationTrack object for color animation
            AnimationTrack colorAnimationTrack =
                    new AnimationTrack(colorKeyframes, AnimationTrack.COLOR);

            //Sequence of keyframes for color transition for light
            //creating color KeyframeSequence object
            //with 2 keyframes and 4 components in each
            //and setting the intepolation type to SLERP
            //that specifies spherical linear interpolation of quaternions
            KeyframeSequence rotationKeyframes =
                    new KeyframeSequence(2, 4, KeyframeSequence.SLERP);
            //this animation will be repeating while animation inside active
            //interval
            rotationKeyframes.setRepeatMode(KeyframeSequence.LOOP);
            rotationKeyframes.setDuration(4000);

            //getRotationQuaternion() method converts vector and angle
            //values to quaternion
            rotationKeyframes.setKeyframe(0, 0,
                getRotationQuaternion(0.0f, new float[] {0.0f, 1.0f, 0.0f}));
            rotationKeyframes.setKeyframe(1, 4000,
                getRotationQuaternion(359.0f, new float[] {0.0f, 1.0f, 0.0f}));

            //AnimationTrack associates a KeyframeSequence with an
            //AnimationController and sets the type of animation.
            //In this case it's orientation animation.
            //creating AnimationTrack object for orientation animation
            AnimationTrack rotationAnimationTrack =
                    new AnimationTrack(rotationKeyframes,
                                       AnimationTrack.ORIENTATION);

            //creating AnimationController which
            //controls the position, speed and weight of an animation sequence
            AnimationController animator = new AnimationController();
            //set controller for animation tracks
            rotationAnimationTrack.setController(animator);
            colorAnimationTrack.setController(animator);

            //set active time interval for controller
            //8000 is not necessarily in milliseconds, it's abstract world
            //time
            animator.setActiveInterval(0, 8000);
            //set speed to half of normal
            //animator.setSpeed(0.5f, 0);
            //Sets a new playback position, relative to world time
            //0 - desired playback position in sequence time units
            //4000 - the world time at which the
            //sequence time must be equal to 0
            animator.setPosition(0, 4000);

            //adding animation tracks to our objects that we
            //wanting to animate
            mesh.addAnimationTrack(rotationAnimationTrack);
            light.addAnimationTrack(colorAnimationTrack);
        }
        /**
         * Calculates the quaternion for a rotation of
         * angle degrees about the vector.
         * @param angle - is an angle of rotation round of vector
         * @param vector - 3D vector of rotation
         * @return quaternion of rotation
         */
        private float[] getRotationQuaternion(float angle, float[] vector) {
            float[] quaternion = new float[4];
            double angleRadians = Math.toRadians(angle)/2.0;
            //i component
            quaternion[0] = vector[0] * (float)Math.sin(angleRadians);
            //j component
            quaternion[1] = vector[1] * (float)Math.sin(angleRadians);
            //k component
            quaternion[2] = vector[2] * (float)Math.sin(angleRadians);
            //w scalar component
            quaternion[3] = (float)Math.cos(angleRadians);
            return quaternion;
        }
    }
}
```

Postconditions

As result we have a MIDlet which draws a rotated cube mesh and an animated color of light source.

(C) Copyright Nokia 2013. All rights reserved.

See also

- Using M3G to draw in Java ME

Supplementary material

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

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.