

```

/**
 *
 */
package com.blindtigersgames.werescrewed.entity;

import static com.badlogic.gdx.graphics.g2d.SpriteBatch.X1;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.X2;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.X3;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.X4;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.Y1;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.Y2;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.Y3;
import static com.badlogic.gdx.graphics.g2d.SpriteBatch.Y4;

import com.badlogic.gdx.Gdx;
// [omitted]
import com.blindtigersgames.werescrewed.graphics.SpriteBatch;

/**
 * A sprite that fills a texture inside a CONVEX polygon. Since this doesn't
 * support origin, you'll want to position your origin point at (0,0) and all
 * other points relative to that. This should behave just like a lib gdx sprite.
 *
 * @author Stew a little bit Kevin :D
 *
 */
public class PolySprite extends Sprite {

    protected Mesh mesh;
    protected ShaderProgram shader;
    protected float[] verts;
    private Array< Vector2 > localVerts;
    private int numVerts;
    protected Rectangle bounds;
    protected Vector2 center;
    private float x, y;
    private float rotation;
    // private Matrix4 modelMat; //holds position and rotation for the polygon.
    // private Matrix4 rotationMat;

    private final int R = 3, G = 4, B = 5, A = 6, U = 7, V = 8, X = 0, Y = 1,
        Z = 2;

    /**
     * Construct a polysprite with a given texture and color

```

```

*
 * @param texture
 *           , size/shape doesn't matter, it will be repeated
 * @param verts
 *           an array of verts, each vector2 is x,y of a vertice in pixels
 * @param r
 *           red color
 * @param g
 *           green
 * @param b
 *           blue
 * @param a
 *           alpha
 */
public PolySprite( Texture texture, Array< Vector2 > verts, float r,
    float g, float b, float a ) {
    super( texture );
    init( verts );
    constructMesh( verts, r, g, b, a );
}

/**
 * Construct a polysprite with a texture and vertices given in pixels. The
 * color will be white.
 *
 * @param texture
 *           texture, size/shape doesn't matter, it will be repeated
 * @param verts
 *           an array of verts, each vector2 is x,y of a vertice in pixels
 */
public PolySprite( Texture texture, Array< Vector2 > verts ) {
    super( texture );
    init( verts );
    constructMesh( verts, 1, 1, 1, 1 );
}

private void init( Array< Vector2 > verts ) {
    this.numVerts = verts.size;
    this.localVerts = verts;
    this.x = 0;
    this.y = 0;
    this.rotation = 0;
    // modelMat= new Matrix4( );
    // rotationMat = new Matrix4( ).rotate( 0, 0, 1, rotation );
}

```

```

@Override
public void setPosition( float x, float y ) {
    translate( x - this.x, y - this.y );
}

/**
 * Sets the x position where the sprite will be drawn. If origin, rotation,
 * or scale are changed, it is slightly more efficient to set the position
 * after those operations. If both position and size are to be changed, it
 * is better to use {@link #setBounds(float, float, float)}.
 */
@Override
public void setX( float x ) {
    translateX( x - this.x );
}

/**
 * Sets the y position where the sprite will be drawn. If origin, rotation,
 * or scale are changed, it is slightly more efficient to set the position
 * after those operations. If both position and size are to be changed, it
 * is better to use {@link #setBounds(float, float, float)}.
 */
@Override
public void setY( float y ) {
    translateY( y - this.y );
}

/**
 * Sets the x position relative to the current position where the sprite
 * will be drawn. If origin, rotation, or scale are changed, it is slightly
 * more efficient to translate after those operations.
 */
@Override
public void translateX( float xAmount ) {
    // this.x += xAmount;
    translate( xAmount, 0 );
}

/**
 * Sets the y position relative to the current position where the sprite
 * will be drawn. If origin, rotation, or scale are changed, it is slightly
 * more efficient to translate after those operations.
 */
@Override

```

```

public void translateY( float yAmount ) {
    // y += yAmount;

    translate( 0, yAmount );
}

/**
 * Sets the position relative to the current position where the sprite will
 * be drawn.
 */
@Override
public void translate( float xAmount, float yAmount ) {
    x += xAmount;
    y += yAmount;

    float[] vertices = this.verts;

    for ( int i = 0; i < numVerts; i++ ) {
        vertices[ 9 * i ] += xAmount;
        vertices[ 9 * i + 1 ] += yAmount;
    }
    mesh.setVertices( verts );
}

/**
 * Set the color, form 0..1
 */
@Override
public void setColor( float r, float g, float b, float a ) {
    int intBits = ( ( int ) ( 255 * a ) << 24 )
        | ( ( int ) ( 255 * b ) << 16 ) | ( ( int ) ( 255 * g ) << 8 )
        | ( ( int ) ( 255 * r ) );
    float color = NumberUtils.intToFloatColor( intBits );
    float[] vertices = this.verts;

    for ( int i = 0; i < numVerts; i++ ) {
        vertices[ 9 * i + R ] = color; // r
        vertices[ 9 * i + G ] = color; // g
        vertices[ 9 * i + B ] = color; // b
        vertices[ 9 * i + A ] = color; // a
    }
}

@Override

```

```

public void setColor( float color ) {
    float[] vertices = this.verts;
    for ( int i = 0; i < numVerts; i++ ) {
        vertices[ 9 * i + R ] = color; // r
        vertices[ 9 * i + G ] = color; // g
        vertices[ 9 * i + B ] = color; // b
        vertices[ 9 * i + A ] = color; // a
    }
}

/**
 * This could be a bug!
 */
@Override
public void setRotation( float degrees ) {
    rotate( degrees - rotation );
    // modelMat.idt( ).rotate( Vector3.Z, rotation ).translate( x, y, 0 );
}

/** Sets the sprite's rotation relative to the current rotation. */
@Override
public void rotate( float degrees ) {
    rotation += degrees;
    // modelMat.idt( ).rotate( Vector3.Z, rotation ).translate( x, y, 0 );

    float cos = MathUtils.cosDeg( rotation );
    float sin = MathUtils.sinDeg( rotation );

    for ( int i = 0; i < numVerts; i++ ) {
        float oldx = localVerts.get( i ).x;
        float oldy = localVerts.get( i ).y;
        verts[ 9 * i + X ] = ( oldx ) * cos - ( oldy ) * sin + this.x;
        verts[ 9 * i + Y ] = ( oldx ) * sin + ( oldy ) * cos + this.y;
    }
    mesh.setVertices( verts );
}

@Override
public Rectangle getBoundingRectangle( ) {
    final float[] vertices = getVertices( );

    float minx = vertices[ X1 ];
    float miny = vertices[ Y1 ];
    float maxx = vertices[ X1 ];
    float maxy = vertices[ Y1 ];
}

```

```

minx = minx > vertices[ X2 ] ? vertices[ X2 ] : minx;
minx = minx > vertices[ X3 ] ? vertices[ X3 ] : minx;
minx = minx > vertices[ X4 ] ? vertices[ X4 ] : minx;

maxx = maxx < vertices[ X2 ] ? vertices[ X2 ] : maxx;
maxx = maxx < vertices[ X3 ] ? vertices[ X3 ] : maxx;
maxx = maxx < vertices[ X4 ] ? vertices[ X4 ] : maxx;

miny = miny > vertices[ Y2 ] ? vertices[ Y2 ] : miny;
miny = miny > vertices[ Y3 ] ? vertices[ Y3 ] : miny;
miny = miny > vertices[ Y4 ] ? vertices[ Y4 ] : miny;

maxy = maxy < vertices[ Y2 ] ? vertices[ Y2 ] : maxy;
maxy = maxy < vertices[ Y3 ] ? vertices[ Y3 ] : maxy;
maxy = maxy < vertices[ Y4 ] ? vertices[ Y4 ] : maxy;

if ( bounds == null )
    bounds = new Rectangle( );
bounds.x = minx;
bounds.y = miny;
bounds.width = maxx - minx;
bounds.height = maxy - miny;

// bounds.x = x-bounds.getWidth( )/2.0f;
// bounds.y = y;

return bounds;
}

@Override
public void draw( SpriteBatch batch ) {
    // this should be called in render()
    if ( mesh == null )
        throw new IllegalStateException(
            "drawMesh called before a mesh has been created." );

    GL20 gl = Gdx.graphics.getGL20( );
    if ( gl != null ) {
        // we don't necessarily need these, but its good practice to enable
        // the things we need. we enable 2d textures and set the active one
        // to 0. we could have multiple textures but we don't need it here.
        gl.glEnable( GL20.GL_TEXTURE_2D );
        gl.glActiveTexture( GL20.GL_TEXTURE0 );
    }
}

```

```

        // setWrap also binds the texture.
        this.getTexture( ).setWrap( Texture.TextureWrap.Repeat,
            Texture.TextureWrap.Repeat );
        // camera * modelview
        mesh.render( shader, GL20.GL_TRIANGLES );
    }

    @Override
    public void setAlpha( float newAlpha ) {
        super.setAlpha( newAlpha );
        for ( int i = 0; i < numVerts; i++ ) {
            verts[ 9 * i + A ] = alpha;
        }
        mesh.setVertices( verts );
    }

    private void constructMesh( Array< Vector2 > verts, float r, float g,
        float b, float a ) {

        shader = WereScrewedGame.defaultShader;

        float minX = Float.MAX_VALUE;
        float minY = Float.MAX_VALUE;
        float maxX = Float.MIN_VALUE;
        float maxY = Float.MIN_VALUE;
        // 9 is 3 positions, 4 colors, and 2 texcoords
        this.verts = new float[ numVerts * 9 ];

        for ( int i = 0; i < numVerts; i++ ) {
            float x = verts.get( i ).x;
            float y = verts.get( i ).y;
            // get the bounds of the poly!
            if ( x < minX ) {
                minX = x;
            } else if ( x > maxX ) {
                maxX = x;
            }
            if ( y < minY ) {
                minY = y;
            } else if ( y > maxY ) {
                maxY = y;
            }
            this.verts[ 9 * i + X ] = x; // x
            this.verts[ 9 * i + Y ] = y; // y

```

```

        this.verts[ 9 * i + Z ] = 0; // z

        this.verts[ 9 * i + R ] = r; // r
        this.verts[ 9 * i + G ] = g; // g
        this.verts[ 9 * i + B ] = b; // b
        this.verts[ 9 * i + A ] = a; // a
    }

    this.bounds = new Rectangle( minX, minY, maxX - minX, maxY - minY );
    center = new Vector2( bounds.x + bounds.width / 2, bounds.y
        + bounds.height / 2 );

    float[ ] texCoords = createTexCoords( verts );

    for ( int i = 0; i < numVerts; i++ ) {
        this.verts[ 9 * i + U ] = texCoords[ 2 * i ]; // u
        this.verts[ 9 * i + V ] = texCoords[ 2 * i + 1 ]; // v
    }
    mesh = new Mesh( true, numVerts, ( numVerts - 2 ) * 3,
        VertexAttribute.Position( ), VertexAttribute.ColorUnpacked( ),
        VertexAttribute.TexCoords( 0 ) );
    mesh.setVertices( this.verts );
    mesh.setIndices( createIndices( ) );
}

/**
 * Creates a triangle fan array of indices for the given vertices
 *
 * @author stew
 * @param numVerts
 * @return
 */
private short[ ] createIndices( ) {
    int numTriangles = numVerts - 2;
    // 3 indices per triangle, (numVerts-2) triangles
    short[ ] indices = new short[ numTriangles * 3 ];
    // insert the first triangle cus it's a shitty mc-weird initial case:
    indices[ 0 ] = 0;
    indices[ 1 ] = 1;
    indices[ 2 ] = 2;
    // then do the rest of the triangles:
    for ( short i = 3; i < numTriangles; ++i ) {
        indices[ i * 3 + X ] = ( short ) ( i + 1 );
        indices[ i * 3 + Y ] = ( short ) ( i + 2 );
        indices[ i * 3 + Z ] = 0;
    }
}

```

```
    }  
    return indices;  
}  
  
int getTextureWidth( ) {  
    return getTexture( ).getWidth( );  
}  
  
int getTextureHeight( ) {  
    return getTexture( ).getHeight( );  
}  
  
/**  
 * really nicely lerp texture coordinates so the texture is not skewed on  
 * the polygon.  
 *  
 * @param verts  
 * @return  
 */  
private float[ ] createTexCoords( Array< Vector2 > verts ) {  
    float[ ] texCoords = new float[ verts.size * 2 ];  
    float texWidth = bounds.width / getTextureWidth( );  
    float texHeight = bounds.height / getTextureHeight( );  
    float halfTexWidth = texWidth / 2;  
    float halfTexHeight = texHeight / 2;  
  
    for ( int i = 0; i < verts.size; ++i ) {  
        texCoords[ 2 * i ] = verts.get( i ).x / bounds.width * texWidth  
            - halfTexWidth;  
        texCoords[ 2 * i + 1 ] = verts.get( i ).y / bounds.height  
            * texHeight - halfTexHeight;  
    }  
    return texCoords;  
}  
}
```