UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Título del Proyecto:

Proyección de elementos 3D sobre imagen de mundo real con librerías basadas en OpenGL

Autor:

Terol Ferrer, David

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles

Septiembre del 2015

Contenido

Introducción

Descripción del problema

Las plataformas actuales disponibles para cargar objetos 3d con OpenGL son generalmente de pago, y la gran mayoría para poder posicionar un objeto dentro del espacio utilizan los denominados targets. Sería interesante descubrir cómo cargar, y posicionar un elemento 3d y posicionarlo en lo referente al observador sin utilizar ningún target.

Objetivos

- Encontrar una librería con la que podamos cargar diseños 3D en varios formatos.
- Acceder a los sensores del dispositivo móvil para posicionarnos.
- Posicionar el objeto en base a la posición real del objeto y la posición del observador.

Motivación

Para el anterior curso de especialización en computación móvil i ubicua, me introduje en el tema de la realidad aumentada y utilizando Unity y targets posicioné un modelo 3D. Pero ahora quiero poder posicionar un modelo son necesidad de un target.

Situación de plataformas Realidad Aumentada Android

Actualmente el uso de aplicaciones de realidad aumentada está muy extendido, se pueden crear apps para:

- **JUEGOS:** Hay muchos juegos que utilizan caras, o dibujos para cargar imágenes o animaciones sobre los targets que proporciona el juego y hacerlo más atractivo.

- **MARKETING Y PUBLICIDAD:** El mundo del marketing se beneficia mucho de esta posibilidad ya que hacer aparecer el producto que se quiere vender generado en 3D sobre el anuncio, hace ver al cliente lo que en realidad es el producto. Es una forma muy atractiva de atraer al cliente.

- **TURISMO:** Por medio de points y de modelos 3d podemos hacer mucho más atractivas las visitas virtuales.
- **EDUCACIÓN:** En educación la realidad aumentada proporciona información adicional al contenido.

Para lograr la realidad aumentada, la gran mayoría utiliza targets para posicionar los elementos.

Actualmente hay diversas plataformas de realidad aumentada disponibles para android:

- Metaio Creator: un programa que permite a los usuarios poder desarrollar contenidos en Realidad Aumentada. No es necesario tener conocimientos de programación, solo tenemos que instalar la aplicación e indicar las plantillas y objetos 3D que estarán relacionados con cada una.

- Blog ingcarlosreina.inkframe.com: Un blog en español que explica como crear aplicaciones de realidad aumentada empezando desde cero.

- aumentaty.com: Otra aplicación que nos ayuda a crear contenido en Realidad Aumentada de forma sencilla e intuitiva. Podéis encontrar un tutorial en español enrecursostic.educacion.es

- unity3d.com + vuforia.com: Aunque con este par de herramientas se pueden hacer muchas cosas cambiando los modelos de los ejemplos existentes, será necesario tener conocimientos de Javascript o C# para Unity para poder personalizar más el resultado.

- Lookar: Framework de Realidad Aumentada para Android. Permite la creación y posicionamiento de formas geométricas pero no de modelos 3d. Puede posicionar por wifi o por gps.

- NYARToolkit: Desarrollada en Japón, es un port de ARToolkit para poder ejecutarla en varias plataformas (móviles y no móviles).

- Min3D: Framework que puede cargar modelos 3D gracias a la implementación de OpenGL.

Para la realización del proyecto he utilizado el framework Min3D. Aunque no cuenta con la posibilidad nativa de cargar las imágenes de background de la cámara del dispositivo ni de geolocalizar un target, nos permitirá estandarizar o hacer más compatible la carga de elementos 3D de cualquier clase y formato de forma nativa.

Arquitectura de la aplicación

Acceso a la cámara

Nativamente el framework Min3D, no permite que se carguen fondos, simplemente objetos. He modifica el código de la librería para poder recoger los frames del preview de la cámara y poder ponerlos de fondo, de modo que el efecto sea más realista.

Mostrar objetos 3D

El código de min3d está basado en OpenGL para Android y es un código apadrinado por Google. Para poder trabajar con min3d lo primero que tenemos que hacer es que nuestra actividad extienda de RenderActivity. De esta manera podremos trabajar con la escena (scene), que consta de la cámara (camera) y de los objetos 3D (childs). Todas las modificaciones de la escena se harán en la clase principal. Así como el acceso a la geolocalización.

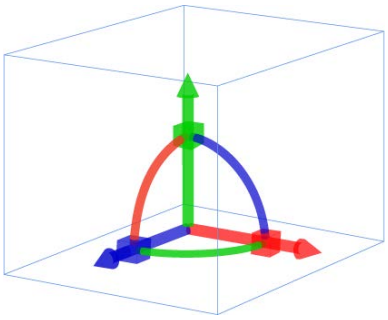
Scene

Scene es algo así como el escenario en el que trabajamos. En él tendremos una cámara que tendrá el papel de observador, y que tendrá una posición y un target, ambas establecidas en un eje x, y, z. El target es hacia el punto que mira la cámara. La posición donde se muestran los objetos vendrá determinada por la distancia y la orientación hacia la que mira el dispositivo. De esta manera, moviéndonos, tendremos diferentes perspectivas de los modelos que tengamos a la vista.

Objetos 3D

Min3D nos proporciona algunos objetos 3D por defecto, en nuestro caso utilizaremos un cubo con las caras de diferentes colores para identificar los puntos de interés creado directamente por OpenGL, pero podríamos mostrar archivos con formato obj o md2 por ejemplo. Cuando un

punto entra dentro del radio de 50m desde el dispositivo, tendremos que crear y situar el objeto 3D en su posición relativa a la nuestra determinando sus coordenadas (x, y, z).



Sensores del teléfono móvil

En nuestra aplicación necesitaremos saber la dirección hacia la que el dispositivo móvil está enfocando. Eso lo conseguiremos gracias a los sensores de los que disponen los dispositivos móviles de hoy en día.

Acceso a los sensores del teléfono móvil

Para acceder al sensor del teléfono móvil Android nos proporciona las clases Android.hardware.SensorManager y Android.hardware.SensorListener. Al implementar en nuestra aplicación el SensorListener tendremos que añadir la función onSensorChanged que es la que se encarga de recibir los datos enviados por los sensores que son solicitados y administrados por la clase SensorManager.

El cambiar el valor de los sensores cuando el móvil se gira se ejecutará el código que hay dentro de onSensorChanged. Debido a que los sensores del teléfono son bastante sensibles, hemos introducido una variable de filtrado para reducir el temblor por la sensibilidad de los sensores, pero aun así se percibe baste temblor.

Hemos utilizado el método onSensorChanged para cambiar la posición del target y de la cámara en función del movimiento.

```
@Override
        public void onSensorChanged(SensorEvent event) {

                // Se comprueba que tipo de sensor está activo en cada momento
    switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            mGravity = event.values;
                                        //cuando los valores del sensor cambian, cambiamos
la orientación
```

```
                                                mAccVals.x = (float) (event.values[1] *
FILTERING_FACTOR + mAccVals.x * (1.0 - FILTERING_FACTOR));
                                                mAccVals.y = (float) (event.values[0] *
FILTERING_FACTOR + mAccVals.y * (1.0 - FILTERING_FACTOR));
                                                mAccVals.z = (float) (event.values[2] *
FILTERING_FACTOR + mAccVals.z * (1.0 - FILTERING_FACTOR));

                                                //Log.d("X", "X: "+mAccVals.x);
                                                //Log.d("Y", "Y: "+mAccVals.y);
                                                //Log.d("Z", "Z: "+mAccVals.z);

                                                scene.camera().target.x = (float)mAccVals.x;
                                        scene.camera().target.y = -(float)mAccVals.z;
                                        //objModel.scale().x = objModel.scale().y = objModel.scale().z
=mAccVals.y;

                                        scene.camera().position.x = -scene.camera().target.x;
                                        scene.camera().position.y = scene.camera().target.y;
                                        //scene.camera().position.z = scene.camera().target.z;
                                        break;
            case Sensor.TYPE_MAGNETIC_FIELD:
                mGeomagnetic = event.values;
                break;
        }


            }
```

Hemos añadido el sensor magnético para posibles ampliaciones del proyecto en cuanto a una mejor colocación del objeto, ya que haciendo uso de la combinación de los dos conseguiríamos una mejor localización del objeto.

Geolocalización

Uno de los problemas de estas coordenadas es que nos devuelven la posición que tenemos en la esfera terrestre, por lo que no nos proporcionará una posición (x,y). Eso es lo que nos interesaría para situar los objetos respecto al dispositivo y por eso decidimos pasar las coordenadas de GPS a el sistema UTM que sí las proporciona en ese formato. El otro problema es la precisión, que depende tanto de la cobertura de los satélites con los que cuenta el sistema, como de otros factores que pueden hacer variar la posición en varios metros.

El acceso a las coordenadas GPS en Android se hace mediante la librería Android.Location y las clases LocationManager y LocationListener. Además necesitaremos añadir al manifest el permiso correspondiente al uso del GPS:

Android.permission.ACCESS_FINE_LOCATION

LocationManager nos proporciona el acceso al servicio de geolocalización y LocationListener se encarga de recibir las notificaciones de LocationManager. Al implementarlo hace falta declarar los métodos onLocationChanged(), onProviderDisabled(), onStatusChanged() y onProviderEnabled().

De los métodos anteriores, utilizaremos el primero para poder actualizar nuestra vista en función de nuestra localización.

También hemos creado un método para poder saber la distancia de un punto a otro respecto a las coordenadas GPS ya que es más fácil utilizar estas coordenadas que no las UTM para calcular la distancia.

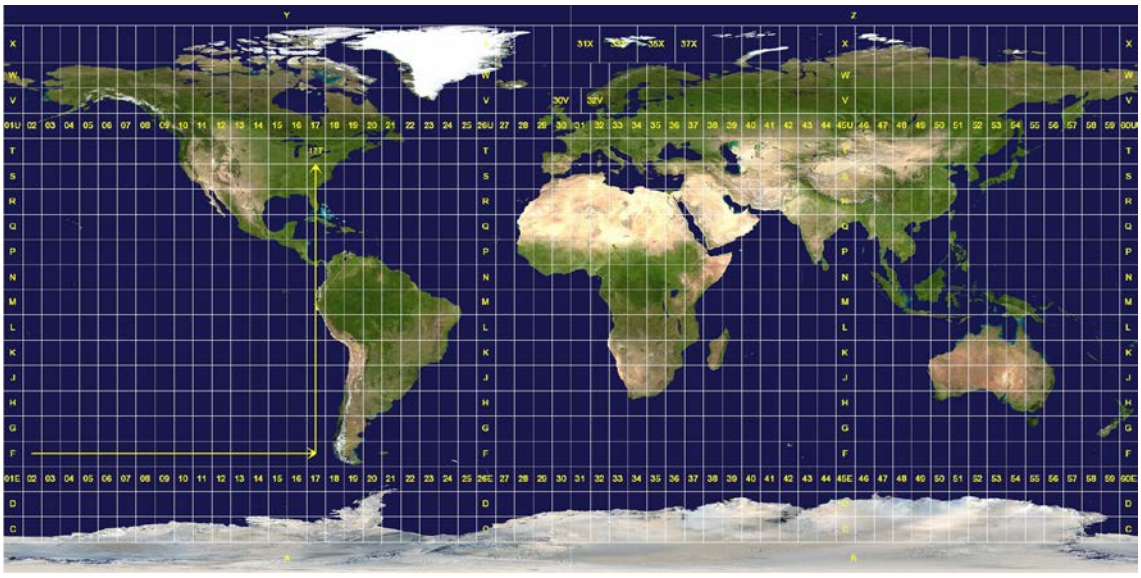
```
//Distancia entre dos puntos

   public double getDistancePoints(double latitude, double longitude){
        double currentLatitude = latitude;
        double currentLongitude = longitude;

        float[] results = new float[3];
        Location.distanceBetween(currentLatitude, currentLongitude,endLatitude,
endLongitude, results);

        BigDecimal bd = new BigDecimal(results[0]);// resultado en metros
        BigDecimal rounded = bd.setScale(2, RoundingMode.HALF_UP);
        double values = rounded.doubleValue();
                return values;
   }
```

Coordenadas UTM

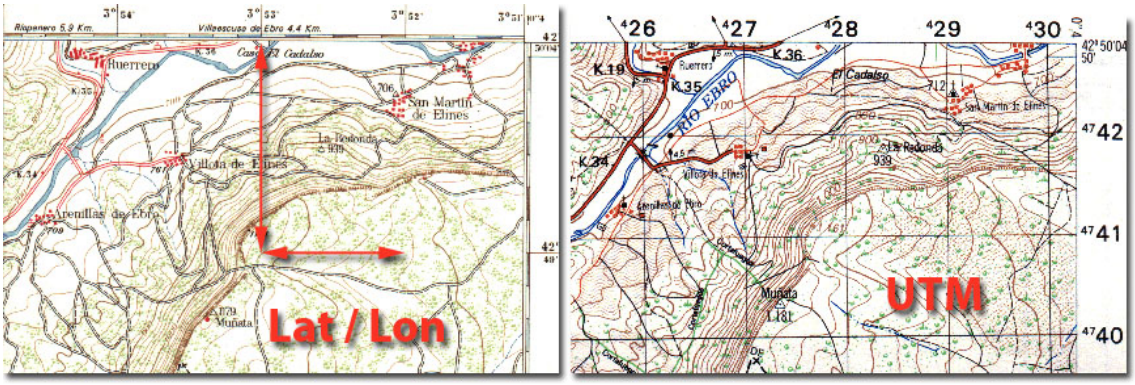
Para trabajar con las coordenadas UTM utilizaremos una clase que podemos obtener de ibm.com, que se llama CoordinateConversion.java y nos proporciona los métodos latLon2UTM y UTMtoLatLong para poder pasar las coordenadas de un modo a otro.

En principio las funciones devuelven un string con todos los parámetros de la conversión, pero como nosotros solo queremos los parámetros referentes a x e y, los demás los hemos desestimado, modificando la clase para que nos devuelva dos coordenadas.

El sistema UTM es un sistema de coordenadas que representa la superficie esférica de la tierra sobre una superficie cilíndrica, tangente al meridiano, que al desplegarse genera un mapa terrestre plano.



Es importante saber, que este sistema localiza la posición dentro de una zona con unas coordenadas x,y , en lugar de un punto. Por lo que cuanto más pequeña sea la zona, mayor será la precisión.



Conclusiones

Es necesario profundizar más en el control de los sensores y de las coordenadas de los puntos UTM, ya que la aplicación posiciona bien los puntos x,y de las coordenadas UTM pero no llega a representar con fiabilidad la altitud del objeto respecto a la cámara.

En cuanto al movimiento de la cámara. Dado que los sensores son muy sensibles, es difícil poder llegar a obtener un control total de la imagen, puesto que el objeto esta en continuo movimiento ya que al actualizarse los valores, la posición se actualiza.

Para el movimiento hemos utilizado el acelerómetro, igual sería más conveniente utilizar otro sensor o la unión de este con otro sensor para poder establecer mejor los giros de cámara y el ratio de escalado del modelo.

En definitiva, a la aplicación todavía le falta un poco para ser del todo utilizable totalmente.

Referencias

Tutoriales sobre algunas aplicaciones que utilizan el GPS:

http://android.javielinux.com/locationgps.php

http://nelopauselli.blogspot.com/2010/08/android-integrando-google-maps-conel.html

http://www.android-spa.com/viewtopic.php?t=1156

Tutoriales para el desarrollo de aplicaciones Android:

http://code.google.com/p/cursopudeandroid/downloads/list

http://developer.android.com/

Paginas de interés de la Realidad Aumentada:

http://es.encydia.com/pt/Realidad_aumentada

Paginas de interés sobre UTM:

http://www.ibm.com/developerworks/java/library/j-coordconvert/index.html

http://www.slideshare.net/profenatu/coordenadas-utm

Página de referencia sobre la librería Min3D:

https://code.google.com/p/min3d/

Clases principales de la aplicación:

Objeto3d.java

```
public class Objeto3d extends RendererActivity implements SensorEventListener, LocationListener {
	private Object3dContainer objModel;
	private SensorManager mSensorManager;
	private Sensor mAccelerometer;
	private Sensor mMagnetometer;
	private Number3d mAccVals;
	private final float FILTERING_FACTOR = 0.7f;
	private int min_metros = 100;

	GPSTracker gps;

	//double endLatitude = 38.694633;
	//double endLongitude = -0.476032;

	double endLatitude = 38.695732;
	double endLongitude = -0.483603;
	double endAltitud = 574.451;

	float utmY = 722561;
	float utmX = 4290696;

	private CoordinateConversion conversor;
	private double[] posUTM;
	private Criteria criteria;
	private LocationManager locationManager;
	Location localiz;
	double distancia;

	float[] mGravity;
	float[] mGeomagnetic;


	// Los angulos del movimiento de la flecha que señala al norte
		float degree;
		// Guarda el valor del azimut
		float azimut;
```

```
                float pitch;
                float roll;


        @Override
        public void onInitScene() {
                // TODO Auto-generated method stub
                super.onInitScene();
                mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mMagnetometer = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
    mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_UI);
    mSensorManager.registerListener(this, mMagnetometer, SensorManager.SENSOR_DELAY_UI);
    mAccVals = new Number3d();
  // create class object
    gps = new GPSTracker(Objeto3d.this);
    mGravity = null;
    mGeomagnetic = null;
    // check if GPS enabled
    if(gps.canGetLocation() == true){

      double latitude = gps.getLatitude();
      double longitude = gps.getLongitude();
      //double altitude = gps.getAltitude();
      double altitude = 580.451;

      float diferenciaAltitud = 0;

      Log.d("GPS latitude", "latitude: "+latitude);
                        Log.d("GPS longitude", "longitude: "+longitude);
      //posicion del observador
                        posUTM = conversor.latLon2UTM(latitude, longitude);
                        //scene.camera().position.x = (float) posUTM[1];
                        //scene.camera().position.y = (float) posUTM[0];
                        //scene.camera().position.z = (float) gps.getAltitude();

                        Log.d("GPS UTM0", "UTM0: "+posUTM[0]);
                        Log.d("GPS UTM1", "UTM1: "+posUTM[1]);


                        posUTM = conversor.latLon2UTM(endLatitude, endLongitude);
      scene.camera().target.x = scene.camera().position.x - (float) posUTM[0];
      scene.camera().target.y = scene.camera().position.y - (float) posUTM[1];
      scene.camera().target.rotateY(90);
      scene.camera().target.rotateX(90);
      distancia = getDistancePoints(latitude,longitude);
      if(altitude > endAltitud){
        diferenciaAltitud = (float)(altitude - endAltitud);
        scene.camera().target.y = scene.camera().target.y - diferenciaAltitud;
      }
      if(altitude < endAltitud){
        diferenciaAltitud = (float)(endAltitud - altitude);
        scene.camera().target.y = scene.camera().target.y + diferenciaAltitud;
      }
      Log.d("altitud resultante", "altitud res: "+diferenciaAltitud);
      Log.d("Distancia", "Distancia: "+distancia);
      Log.d("target", "X: "+scene.camera().target.x);
      Log.d("target", "Y: "+scene.camera().target.y);


      objModel.scale().x = (float)(1/distancia);
      objModel.scale().y = (float)(1/distancia);
      objModel.scale().z = (float)(1/distancia)*.2f;
      //scene.camera().target.z = scene.camera().position.z;
    }else{
      gps.showSettingsAlert();
    }

        }

//Distancia entre dos puntos

  public double getDistancePoints(double latitude, double longitude){
```

```
        double currentLatitude = latitude;
        double currentLongitude = longitude;

        float[] results = new float[3];
        Location.distanceBetween(currentLatitude, currentLongitude,endLatitude, endLongitude, results);

        BigDecimal bd = new BigDecimal(results[0]);// resultado en metros
        BigDecimal rounded = bd.setScale(2, RoundingMode.HALF_UP);
        double values = rounded.doubleValue();
                return values;
  }



        @Override
        public void initScene() {
                scene.lights().add(new Light());
            scene.lights().add(new Light());
            scene.backgroundColor().setAll(0,0,0,0);
            Light myLight = new Light();
            myLight.position.setZ(150);
            scene.lights().add(myLight);
                IParser parser = Parser.createParser(Parser.Type.OBJ, getResources(),
"com.example.modelo3dmin:raw/casa_obj", true);
                parser.parse();
                scene.backgroundColor().setAll(0x00);
                conversor = new CoordinateConversion();
                //scene.camera().target.x = utmX;
                //scene.camera().target.y = utmY;
                        if(distancia <= 100){
                                objModel = parser.getParsedObject();
                                //objModel.scale().x = objModel.scale().y = objModel.scale().z = 0.3f;
                                scene.addChild(objModel);
                        }

        }
        @Override
        public void updateScene() {

                //objModel.rotation().x++;
                //objModel.rotation().z++;
        }
        @Override
        public void onSensorChanged(SensorEvent event) {

                // Se comprueba que tipo de sensor está activo en cada momento
    switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            mGravity = event.values;
                                                //cuando los valores del sensor cambian, cambiamos la
orientación

                                                mAccVals.x = (float) (event.values[1] *
FILTERING_FACTOR + mAccVals.x * (1.0 - FILTERING_FACTOR));
                                                mAccVals.y = (float) (event.values[0] *
FILTERING_FACTOR + mAccVals.y * (1.0 - FILTERING_FACTOR));
                                                mAccVals.z = (float) (event.values[2] *
FILTERING_FACTOR + mAccVals.z * (1.0 - FILTERING_FACTOR));

                                                //Log.d("X", "X: "+mAccVals.x);
                                                //Log.d("Y", "Y: "+mAccVals.y);
                                                //Log.d("Z", "Z: "+mAccVals.z);

                                                scene.camera().target.x = (float)mAccVals.x;
                                        scene.camera().target.y = -(float)mAccVals.z;
                                        //objModel.scale().x = objModel.scale().y = objModel.scale().z
=mAccVals.y;

                                        scene.camera().position.x = -scene.camera().target.x;
                                         scene.camera().position.y = scene.camera().target.y;
                                         //scene.camera().position.z = scene.camera().target.z;
                                         break;
        case Sensor.TYPE_MAGNETIC_FIELD:
```

```
            mGeomagnetic = event.values;
            break;
        }


        }



        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
                // TODO Auto-generated method stub

        }
        @Override
        public void onLocationChanged(Location location) {
                // TODO Auto-generated method stub
                //Al cambiar la localización actualizamos la ubicación de la cámara
                //posicion del observador
                //if(distancia <= 1000){
                        float diferenciaAltitud = 0;
                        double altitude = 580.451;

                        posUTM = conversor.latLon2UTM(location.getLatitude(),location.getLongitude());
                        Log.d("GPS location PosUTM1", "Pos: "+posUTM[1]);
                        Log.d("GPS location PosUTMo", "Pos: "+posUTM[0]);
                        distancia = getDistancePoints(location.getLatitude(),location.getLongitude());
                        objModel.scale().x = (float)(1/distancia);
        objModel.scale().y = (float)(1/distancia);
        objModel.scale().z = (float)(1/distancia)*.2f;
        if(altitude > endAltitud){
            diferenciaAltitud = (float)(altitude - endAltitud);
            scene.camera().target.y = scene.camera().target.y- diferenciaAltitud;
        }
        if(altitude < endAltitud){
            diferenciaAltitud = (float)(endAltitud - altitude);
            scene.camera().target.y= scene.camera().target.y + diferenciaAltitud;
        }
        scene.camera().position.y = scene.camera().target.y;
                        posUTM = conversor.latLon2UTM(endLatitude, endLongitude);

        //          }


        }
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
                // TODO Auto-generated method stub

        }
        @Override
        public void onProviderEnabled(String provider) {
                // TODO Auto-generated method stub

        }
        @Override
        public void onProviderDisabled(String provider) {
                // TODO Auto-generated method stub

        }

}
```