UNIVERSITAT OBERTA DE CATALUNYA

Enginyeria Informàtica

Visualització 4D a través de mòbils Android

Alumne/a: Òscar Estévez Soler

Dirigit per: Laia Descamps Vila

Co-dirigit per:

CURS 2011-2012 (Febrer/Setembre)

Universitat Oberta de Catalunya

Aula 1: Sistemes d'informació geogràfica
2on semestre curs 2011-2012.
Alumne: Òscar Estévez Soler

Visualització 4D a través de mòbils Android

05.032-TFC-SIG
Curs 2011-12. 2on Semestre

Universitat Oberta de Catalunya

Òscar Estévez Soler

Universitat Oberta de Catalunya

Aula 1: Sistemes d'informació geogràfica
2on semestre curs 2011-2012.
Alumne: Òscar Estévez Soler

Dedicatòria

A la petita Ia, el seu somriure m'encoratja
a continuar esforçant-me cada dia.

A la Nuri, amb tu tinc encetat el millor
projecte de la meva vida.

ÍNDEX

1. Agraïments

Són moltes les persones que han influenciat en mi i la meva carrera durant els anys, moltes, i sovint he trigat anys en adonar-me'n. Des d'aquells fantàstics professors del col·legi Vall-vera de Salt que tant em mimàvem, més tard seguint el camí dels meus germans estudiant a l'IES Salvador Espriu on els docents vetllaven per la nostre formació i superació. Després van arribar els durs anys a la UPC on vaig aprendre el que era esforçar-se de valent amb els professors més exigents i capacitats que havia vist mai, el tracte familiar de la UDG i per acabar la professionalitat i el bon fer del personal de la UOC. Agraeixo l'esforç de tots els professionals de l'ensenyament que m'han ajudat.

No tinc prou paraules per agrair tot el que la meva família ha fet per mi. Els meus pares; que mai m'han dit el que haig de fer però sempre han recolzat les meves decisions, amb la seva discreció i saber fer m'han sabut portar per el bon camí. Els meus germans: Txus i David són les millors influencies que podria desitjar i mai voldria perdre del meu costat.

A la Laia Descamps que amb els seus comentaris i ajudes ha aconseguit fer d'una experiència esgotadora un repte amb una gran satisfacció final.

2. Introducció

Aquest projecte de final de carrera tracta del sistemes d'informació geogràfica (a partir d'ara SIG) i en concret del seu ús amb les noves tecnologies en telefonia mòbil. No aprofundirem sobre els SIG en general, sinó que ens centrarem a la tecnologia mòbil i el desenvolupament d'una demostració d'una aplicació de visió 4D.

2.1. Introducció als SIG aplicats a la telefonia mòbil

Els **Sistemes d'Informació Geogràfica** (SIG, de l'anglès *GIS – Geographic Information System*), són una combinació de maquinari, programari i dades geogràfiques que serveix per resoldre problemes de gestió geogràfica (gestió de flotes, gestió de rutes, representació de mapes, etc.). L'evolució d'aquests ha estat altament influenciada pel progrés de les tecnologies d'informació i ha estat paral·lela a l'evolució de la informàtica, des dels SIG basats en ordinadors centrals (*mainframe*) passant per els SIG d'escriptori (*desktop*) i fins arribar als SIG distribuïts (*distributed*). Veure figura 1.

Ens centrarem en els SIG distribuïts que són aquells programes que s'executen a internet o en un entorn amb xarxa sense fils, si s'executen a internet estarem parlant de Internet SIG (o Web SIG) i si és amb xarxa sense fils de **SIG mòbils**. Aquesta tecnologia ha estat possible gràcies al recent desenvolupament de Internet i les comunicacions sense fils, recolzada per una ràpida expansió de la xarxa a baix cost i la generalització de PC's i dispositius mòbils amb capacitat de navegar per Internet.

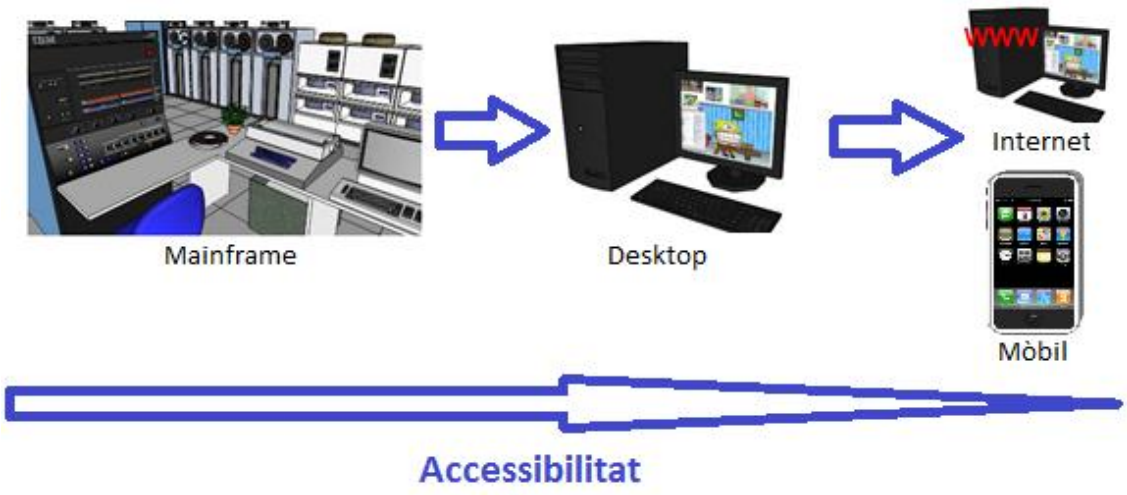

Figura 1 (Evolució dels SIG)

Els SIG distribuïts es basen en el concepte client/servidor: el client fa peticions de dades o càlculs al servidor, i aquest últim lliura el resultat al client. Les connexions entre client i servidor s'estableixen mitjançant un protocol de comunicació, normalment TCP/IP. Els SIG distribuïts gaudeixen de la gran avantatge d'Internet, que és un distribuïdor gegant de sistemes on poden residir els SIG en diferents ordinadors a la xarxa, i permet que els clients puguin accedir des de qualsevol punt d'aquesta (actualment a gairebé tot el mon).

Els SIG mòbils pateixen algunes limitacions causades pels dispositius on són executats: pantalles de petites dimensions, capacitat d'emmagatzematge limitat i una bateria amb autonomia limitada. Tot i que podem dividir els dispositius en dos classes segons el tipus de client: els lleugers, amb poca capacitat de càlcul on el servidor executa la majoria d'accions, o pesants, on el client te capacitat de càlcul i descarrega el servidor d'algunes tasques.

Com hem comentat anteriorment, els SIG ja s'apliquen a la telefonia mòbil gràcies a la gran evolució d'aquest tipus de dispositiu i això ha provocat l'expansió dels **serveis basats en la localització** (*LBS* de l'anglès *Located Based Services*). Aquests serveis necessiten conèixer la posició del dispositiu (es basen en els SIG) per donar diferents funcionalitats a l'usuari: web 2.0, localització de POI's pròxims, càlcul de rutes, etc. És un concepte existent des de l'any 2.000 però és amb el llançament de dispositius mòbils amb capacitats per suportar LBS (iPhone i Android) quan el número d'aplicacions creix de forma exponencial.

A continuació (taula 1) mostrarem els fets més significatius en l'evolució dels SIG i la tecnologia mòbil.

Any	Esdeveniment
1982	Es funda **Sun Microsystems** per part de *Vinod Khosla*, *Andy Bechtolsheim*, and *Scott McNealy*
1991	El primer kernel **Linux** és distribuït per *Linus Torvalds*
1992	**IBM** presenta **Simon** • És considerat el primer *smartphone* • Incorporava un calendari, llibre d'adreces, rellotge mundial, calculadora, bloc de notes, jocs i podia enviar i rebre fax. • Pantalla tàctil, sense cap botó. • Teclat predictiu.
1995	Es distribueix **Java** per par de **Sun Microsystems**
1996	**Palm** presenta **Pilot 100** creat per *Jeff Hawkings* • El primer PDA amb èxit • Es beneficia dels primers intents errats per creat una plataforma de mà per part de Go i Apple • Incorporava un software de reconeixement d'escriptura molt avançat.
1998	Es funda **Google** per part de *Larry Page* y *Sergey Brin*
2003	Es funda **Where 2 technologies** per els germans *Lars* i *Jens Rasmussen*, *Noel Gordon* i *Stephen Ma*
2004	1. **Google** compra **Where 2 technologies** per ser la base de **Google Maps** 2. **Google** compra **Keyhole** que desenvolupa **Earth Viewer**
2005	1. Apareix el terme **Mashup** per combinar en una sola pàgina web varies funcionalitats de diferents orígens 2. Apareix **AJAX**, com a tècnica de desenvolupament per crear pàgines web interactives 3. **Google Maps** ja està disponible a la web 4. **Google** compra l'empresa **Android Inc**. i continua el desenvolupament del sistema operatiu **Android**.
2006	1. **Google Earth** ja està disponible per els usuaris. • És una evolució del **Earth Viewer** • Fa servir les dades recollides per els models digitals d'elevació (DEM) de la **NASA**. 2. **Google** Introdueix **Google Maps** per mòbils, un intent d'aplicació que pugui ser executada sobre un dispositiu mòbil basat en **Java**. 3. **Microsoft** llança al mercat **Virtual Earth**, com a resposta a **Google Maps**
2007	**Apple** comercialitza l'**iPhone**

	• Incorpora de connexió a internet. • Pantalla tàctil • Inclou càmera, e-mail, Wi-Fi, etc.
2008	1. Apareix **Android 1.0** 2. **HTC** comercialitza el primer mòbil amb sistema operatiu **Android**, el **HTC Dream** 3. **Google Maps** està disponible per les següents plataformes • Android • iOS (iPhone, iPad) • Windows Mobile • Symbian • BlackBerry • Palm OS 4. **Android** ja representa el 2.8 % dels *smartphones* mundials
2009	1. Apareixen **Android 1.5** (*Cupcake*), **Android 1.6** (*Donut*) i **Android 2.0** (*Eclair*) 2. **Oracle** compra **Sun Microsystems**
2010	1. Apareixen **Android 2.2** (*Froyo*), **Android 2.**3 (*GingerBread*) 2. **Google** amb col·laboració amb **HTC** comercialitza el seu primer terminal mòbil el **Nexus One** 3. **Android** ja supera en ventes a **iPhone**
2011	1. Apareix **Android 3.0** (*Honeycomb*) la primera versió optimitzada per *tablets* i **Android 4.0** (*Ice cream sandwich*) 2. Disponible **arcGis** de **ESRI** per **Android** i **iOS**
2012	La xifra de telèfons **Android** venuts supera els 300 milions

Taula 1 (evolució dels SIG i telefonia mòbil)

Es pot comprovar que en els darrers 10 anys l'evolució ha estat frenètica.

2.2. Introducció al PFC

En l'apartat anterior hem explicat el que és un SIG mòbil i la seva evolució tecnològica. Aquest PFC tracta d'aprofitar tot aquesta tecnologia, que a dia d'avui està disponible per tothom a un cost relativament baix, i investigar sobre ella per desenvolupar una aplicació demostració. A l'enunciat d'aquest PFC es definia exactament la tecnologia a utilitzar: el sistema operatiu **Android** i un dispositiu *smartphone* (tot i que no és obligatori disposar-ne d'un) o un emulador; són les bases amb les quals ens treballarem

Desenvoluparem una aplicació que ofereixi a l'usuari una visita en el temps d'un monument determinat. Amb les tecnologies de posicionament del GPS o mitjançant el 3G, i les aplicacions de realitat augmentada detectarem quin és el monument visitat i la nostra aplicació oferirà imatges d'aquest en diferents moments històrics. Resumint, es tracta d'oferir un viatge en el temps a través de la història del monument que l'usuari estigui visitant en aquell moment.

Per dur a terme l'objectiu final desenvoluparem tot un seguit de tasques.

- Aprofundir en el coneixement i aplicació dels SIG i sobretot en SIG mòbils.

- Aprofundir en el sistema Android i les seves eines de programació.

    - Fer recerca del millor entorn de programació.
    - Instal·lar i configurar aquest entorn.
    - Fer recerca de modes d'emulació de proves.
    - Fer recerca de càrrega d'aplicacions en un smartphone real.
    - Fer recerca d'eines de geoposicionament i instal·lar-les.
    - Fer recerca d'informació sobre *frameworks* de realitat augmentada, i la seva posterior instal·lació.
    - Estudiar l'emmagatzematge de dades per Android.
    - Finalment, desenvolupament de l'aplicació de demostració amb els components seleccionats en els punts anterior.

- Seleccionar un punt d'interès i obtenir-ne imatges en diferents moments històrics. Aquest punt d'interès ha de tenir una rellevància històrica notable per tal de poder disposar d'un bon ventall d'imatges i, per facilitat, ha de ser pròxim al nostre lloc de treball.

3. Eines de desenvolupament

A l'enunciat d'aquest projecte, s'especificava clarament que l'aplicació ha de ser desenvolupada per dispositius amb sistema operatiu **Android**, això ens marca el camí a seguir en l'elecció de les eines que podem fer servir.

Per el desenvolupament de la nostra aplicació de demostració treballarem amb una màquina virtual (a partir d'ara MV) creada amb *VMware Player 4.0.2*. S'han tingut en compte les recomanacions de Google per definir les característiques de la màquina, que són les següents:

- Windows XP professional 32 bits amb SP3.
- 2 Gb de memòria RAM.
- 20 Gb de disc dur.
- 2 Processadors.
- S'han compartit carpetes amb entre l'equip Host i la MV.

L'elecció del sistema operatiu, ha estat basada en l'experiència que tenim del mateix. Podríem haver triat diverses distribucions Linux, però el desconeixement de les mateixes hagués impedit un desenvolupament àgil, i no està dintre de l'abast d'aquest projecte.

La possibilitat de treballar amb MV, ens permetrà poder instal·lar diversos programes i fer proves amb ells sense malmetre el nostre entorn de treball. El treballar amb *VMware* és una elecció basada en l'experiència amb les eines d'aquest fabricant.

Les proves amb un dispositiu mòbil, les farem sobre un smartphone model *Samsung Galaxy SII* amb sistema operatiu Android v. 2.3.5 (GingerBread), tot i que no és obligatori disposar d'aquest mòbil gaudim de l'oportunitat de fer-ho i ho aprofitarem.

Hem instal·lat els divers del telèfon (*SAMSUNG USB Driver for Mobile Phones v. 1.4.103.0*), i hem habilitat la possibilitat de que la MV detecti els USB connectats a la màquina host.

El maquinari està definit i configurat, a continuació estudiarem les eines de programació necessàries per el desenvolupament del projecte.

3.1. Eclipse

El sistema operatiu Android està basat en un nucli Linux però dissenyat específicament per telèfons mòbils. L'estructura d'aquest sistema operatiu es composa d'aplicacions que executen el seu propi procés en un *framework* Java en una instància de màquina virtual *Dalvik* (és una MV optimitzada per l'execució minimitzant l'ús de memòria i fa córrer classes compilades per un compilador Java). Com tots sabem la màquina virtual Java sempre és necessària ja que es tracta d'un llenguatge interpretat.

Malgrat tot, amb el llenguatge de programació no n'hi ha prou per començar a desenvolupar aplicacions, el fabricant posa a la nostra disposició l'**Android SDK**, el conjunt d'eines de software que ens permetran programar, configurar l'accés per proves a un dispositiu, tutorials, documentació, etc. Aquest paquet està escrit en Java.

Fins ara hem vist que tant sistema operatiu com les utilitats de programació estan basades en Java. És evident doncs, que seleccionarem un entorn de programació basat en aquest llenguatge.

L'entorn de desenvolupament integrat (IDE) oficial suportat per ***Android*** és **Eclipse** i per tant ens decantem per aquesta opció. Val a dir, que Eclipse és gratuït.

Hi ha altres alternatives, que enumerem a continuació a mode informatiu:

- **Netbeans**: és també un IDE per a Java. El plugin Android no és oficial.
- **Appcelerator Titanium**: codi Javascript, multi plataforma iOS i Android. Ràpid desenvolupament, però problemes amb la gestió de memòria.
- **Html5**: ofereix moltes possibilitats en el desenvolupament web. No és tan òptim com el codi natiu Android.

- **Ruboto**: permet executar scripts escrits en Ruby a Android.
- **Basic4Android**: codi semblant a Visual Basic. Entorn gràfic molt agradable, no permet tota la potencia del codi natiu Android.
- **MonoDroid**: codi C# i .NET. És de pagament.
- **Lazarus Free Pascal**: Projecte de codi lliure que vol fer un compilador en Pascal per Android. En procés de desenvolupament.

Fins i tot ja estan apareixent els IDE de programació dintre del propi Android, per exemple AIDE (*Java IDE for Android*).

La instal·lació d'un IDE no és pas obligatòria, ja que podríem fer servir les eines de la línia de comandes, però un IDE sempre aporta més productivitat. També existeix la possibilitat de programar en codi C directament sobre el nucli d'Android mitjançant l' **Android NDK** (*Native Development Kit*), però no és necessari en el nostre cas ja que no creiem que patim cap limitació per part de les API's existents i que farem servir en el nostre desenvolupament.

El primer pas per la configuració del nostre entorn de treball és instal·lar el JDK de Java, Android recomana els JDK 5 o 6. Per fer-ho només cal anar a la pàgina web de Sun (http://www.oracle.com/technetwork/java/javase/downloads/index.html) i descarregar la versió que necessitem, en el nostre cas la 1.6.0-31. La instal·lació és senzilla i es fa sense cap mena de problema. Per comprovar la correcte instal·lació, només cal escriure la comanda *java –version* a la línia de comandes, i obtenir un resultat semblant al que es mostra a la figura 2.

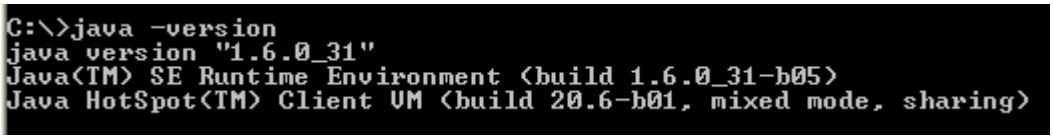

Figura 2 (versió de Java instal·lada)

El següent pas, és instal·lar l'IDE Eclipse. Android té com a únic requeriment treballar, com a mínim, amb la versió 3.3.1, però és recomanable treballar amb la més actual. Descarregarem el programa de la pàgina d'Eclipse i de la secció de *Eclipse IDE for Java depevolers* (http://www.eclipse.org/downloads/). Un cop descarreguem el paquet només cal extreure'l en el que serà el nostre directori de treball definitiu *C:\eclipse*.

3.2. Android SDK

Anteriorment ja hem comentat que només amb Java no és possible programar per un entorn Android, ens cal quelcom més, ens cal el Android SDK. El SDK és el conjunt d'eines i API's necessàries per començar a desenvolupar a la plataforma Android fent servir programació Java. Les eines més important incloses en aquest paquet són:

- **Android SDK Manager**: permet instal·lar i gestionar les actualitzacions del nostre entorn de desenvolupament.
- **AVD Manager**: permet la gestió de les diferents AVD (Android Virtual Device) .
- **Emulator**: permet l'execució d'aplicacions sense disposar d'un dispositiu físic.
- **Dalvik Debug Monitor Server (ddms)**: permet debugar les aplicacions Android.

Hi ha moltes més utilitats disponibles, que mostrem a mode informatiu.

- **Dmtracedump**: genera un diagrama de les crides a la pila, per fer seguiment dels fitxers de log.
- **Draw-9-patch**: permet crear un **NinePatch** gràfic fent server tecnologia WYSIWYG.
- **Hierachy Viewer (hierachyviewer)**: Permet depurar i optimitzar la interfície d'usuari.
- **Hprof-conv**: converteix el fitxer HPROF (generat per l'Android SDK) a un format estàndard, el què permet la seva visualització.
- **Layoutopt**: permet un anàlisis ràpid dels layouts de l'aplicació per optimitzar la seva eficiència.
- **Mksdcard**: permet crear una imatge de disc que es pot fer servir amb l'emulator, per simular un una SD card.(o semblant).
- **Monkey**: genera events d'usuari com clicks, gestos, etc, per fer proves d'estrès de la nostra aplicació.
- **Monkeyrunner**: Proveeix d'una API que permet programar aplicacions que controlin un dispositiu Android o emulador fora de codi font Android.
- **ProGuard**: permet minimitzar i optimitzar el codi, esborrant codi sobrant.
- **Sqlite3**: permet l'accés als fitxers SQLite.

- **Traceview**: permet una visió gràfica del log d'execució de l'aplicació.
- **Zipalign**: optimitza els fitxers .apk.

Per instal·lar aquest paquet, l'hem de descarregar de la pàgina de descàrregues d'Android (http://developer.android.com/sdk/index.html). És un fitxer .ZIP que conté un directori *android-sdk-windows* el qual el col·locarem dintre del directori *C:\google*.

El següent pas és descarregar els components SDK. Executarem el programa *SDK manager.exe* i veurem un llistat de tots els paquets disponibles. Seleccionarem la casella *Accept All* i a continuació *Install.*

Finalment, Google ha desenvolupat un complement per Eclipse que s'anomena **Android Development Toolkit** (ADT) per fer més fàcil la programació des d'aquest entorn. L'instal·larem, des d'Eclipse, i indicant la URL d'actualització de les ADT (http://dl-ssl.google.com/android/eclipse/).

Arribats a aquest punt ja tenim les eines bàsiques per desenvolupar.

3.3. Framework de realitat augmentada

La realitat augmentada (d'ara endavant AR) és la tecnologia que permet crear una realitat que és mescla d'una visió directa d'un entorn físic real amb elements virtuals. Nosaltres la farem servir per la visualització dels diferents moments en el temps del monument visitat. A la figura 3 veiem un exemple de funcionament d'AR.

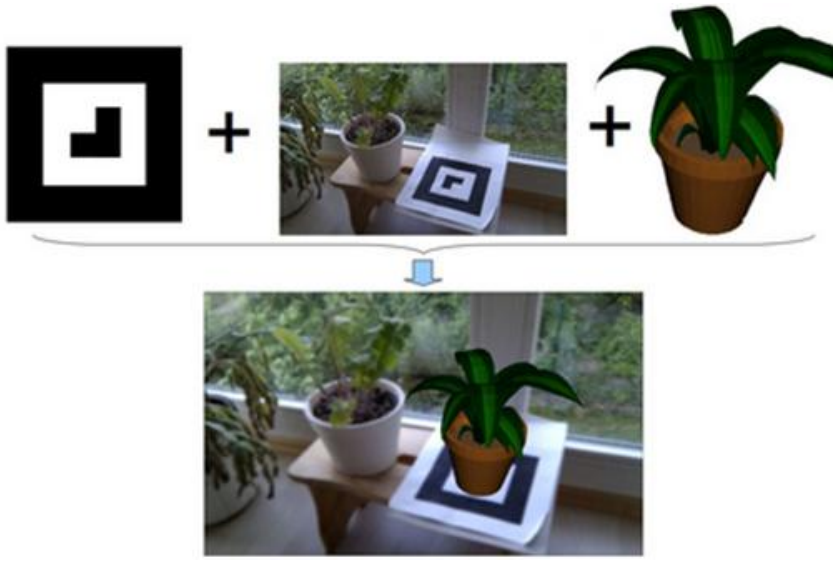

Figura 3 (Exemple de funcionament de AR)

Un sistema AR segons *Ronald Azuma* [1] ha de tenir les següents característiques:

- Combina elements reals i virtuals: si no conté un dels dos no estem parlant de AR.
- És interactiva en temps real: si no hi ha interacció, no és AR. Una pel·lícula amb elements virtuals no permet interacció i per tant no la considerem AR.
- Està registrada en 3D: ha de ser tridimensional i no bidimensional.

I només necessita de 3 components, una càmera per veure l'entorn real, una pantalla per mostrar el resultat i un dispositiu capaç de fer la fusió de la realitat amb els elements virtuals.

Les actuals aplicacions d'AR, es poden dividir en dos grans grups segons com resolen la posició de la càmera respecte l'objecte virtual. El primer grups, són les aplicacions que treballen amb una estratègia basada en marques, una mena d'etiqueta en el món real que indica la posició que ha de tenir l'objecte virtual. Aquesta solució és òptima a nivell de sistema ja que consumeix pocs recursos, però amb l'inconvenient de que les marques poder ser molestes (Figura 4). El segon grup, engloba les aplicacions que detecten la posició mitjançant les capacitats del dispositiu (GPS, 3G, etc.), es a dir, que representen les escenes amb ajuda d'informació geogràfica.

Figura 4 (exemple de AR que fa servir marques)

Per Android hi ha un gran ventall de frameworks per desenvolupar aplicacions AR. Tot seguit exposem una petita llista de les més rellevants:

- **andAR**: segueix una estratègia de representació sobre marques físiques. És d'ús lliure.
- **mixare**: segueix l'estratègia de posicionament per GPS, però està més enfocat a la representació de caixes de texts i imatges 2D.
- **Layar**: segueix l'estratègia de posicionament per GPS. Llicencia de pagament i tancada, sense possibilitat d'accedir al codi font.
- **LookAR**: es troba en fase beta. Neix per millorar les mancances de andAr i mixare en posicionament en interiors, segueix l'estratègia de posicionament via GPS. Llicencia oberta.
- **DroidAR**: segueix l'estratègia de representació mitjançant posició. Llicencia oberta.
- **QCAR - Qualcomm Augmented Reality SDK**: actualment anomenat *Vuforia*. Segueix l'estratègia de marques.
- **Popcode**: actualment sense manteniment.

Per el nostre desenvolupament, les solucions que segueixen l'estratègia de marques no són bones, ja que descartem la idea de posar marques en el monument a tractar i sobretot pensat que podem desenvolupar una AR basant-nos en el geoposicionament, aquesta segona estratègia encaixa millor en la definició d'aquest PFC. D'igual manera descartem aquelles solucions que tinguin una llicencia de pagament, pensant en una possible evolució d'aquest PFC i evidentment en el cost de desenvolupament del mateix. Per tant, la nostra elecció estarà entre mixare, LookAR i droidAD. Mixare la descartem per les seves limitacions, LookAR per ser una versió Beta, droidAR és aparentment la opció més vàlida:

- Versió estable.
- Codi obert.
- Gran quantitat d'informació a la Web, documentació, tutorials, vídeos, etc.
- Estratègia de posicionament GPS.

El següent pas és instal·lar el necessari per poder desenvolupar amb droiAR. Ens ha sorprès la facilitat d'instal·lació d'aquesta eina, simplement des del nostre Eclipse, i amb el plugin d'exploradors de repositoris SVN (que hem tingut que instal·lar

prèviament), introduint la URL del projecte http://droidar.googlecode.com/svn, ja podem començar a desenvolupar. Hem seguit el tutorial per fer la nostra primera aplicació, que es tractava de fer aparèixer un parell d'objectes davant nostre a uns 10 metres, i el resultat ha estat francament satisfactori. A les figures 5 i 6 veiem els resultat, es pot comprovar que no hi ha la distancia de 10 metres, però les fotos s'han pres en un interior i allà el geoposicionament no és del tot acurat.

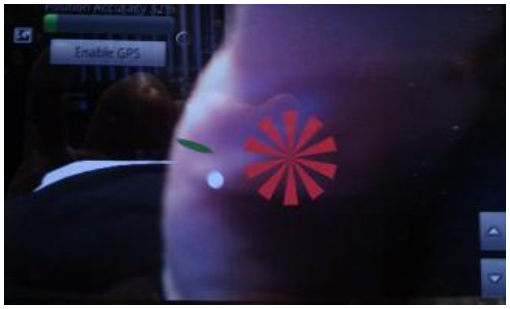

Figura 5 (Resultat tutorial droidAR I)

Figura 6 (Resultat tutorial droidAR II)

3.4. Base de dades

Un dels grans problemes que se'ns plantejava era l'emmagatzematge de les imatges del nostre monument "demo", hem de ser capaços de guardar diverses imatges del monuments en diferents moments històrics, i hem de pensar que en un futur haurem d'emmagatzemar imatges de més d'un indret i totes elles han de ser ràpidament accessibles des de la nostra aplicació. Tenim vàries solucions:

- Emmagatzemar les imatges a l'estructura de fitxers: Això representa crear una organització pròpia, i implementar una gestió de la mateixa. Les dades sempre viatjarien amb el dispositiu, i tindríem un accés ràpid.
- Treballar amb una base dades (també les anomenarem BD): Aprofitar algun motor de base de dades per estalviar-nos la implementació de l'estructura, aprofitaríem tots els seus avantatges.

La idea de treballar amb una base de dades per a l'emmagatzematge de les imatges, sembla la més encertada. Una de les premisses a considerar és que l'accés a les imatges ha de ser ràpid, i per tant les dades estaran localment al dispositiu (així no dependrem de connexions lentes, ni de falta de cobertura) , conseqüentment descartem crear un servidor remot de bases de dades (l'evolució de l'aplicació possiblement ho necessiti, però nosaltres en aquest projecte no ho tractarem).

A l'apartat 3.2 Android SDK, vam veure que unes de les eines que apareix en el SDK és el Sqlite3 que ens dona accés a bases de dades SQLite. El sistema gestor de bases de dades SQLite és un sistema transaccional i està dissenyat per treballar sense servidor. Les bases de dades SQLite s'incrusten a l'aplicació principal aconseguint així un temps de resposta molt ràpid. Aquestes bases de dades són molt interessant per el nostre projecte ja que permeten camps de tipus *BLOB* que ens permetran emmagatzemar imatges.

L'elecció de SQLite com a la base de dades pel nostre projecte, ha estat una decisió senzilla: compleix amb les nostres necessitats i és totalment accessible des d'aplicacions Android. Hem investigat de manera superficial alternatives com *MySQL*, però el nivell de complexitat era més elevat, i no és un dels objectius finals d'aquest treball.

Els passos seguits per instal·lar el SQLite en el nostre maquinari són: descarregar el fitxer *sqlite-dll-win32-x86-3071100.ZIP* de la pàgina del fabricant http://www.sqlite.org/download.html i descomprimir-lo a la ruta *C:\sqlite*. El que hem fet en aquest pas, ha estat instal·lar un petit executable que ens permet executar consultes SQL des de la línia de comandes (Figura 7).

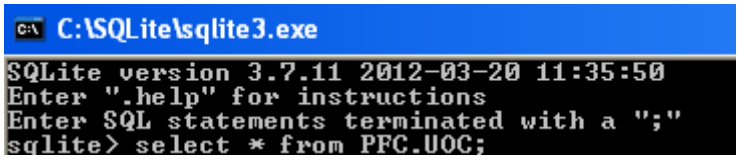

Figura 7 (SQLite)

Malgrat no és obligatori per treballar amb SQLite, hem optat per cercar un IDE gràfic (Figura 8) per a la gestió de les bases de dades, *SQLite Administrator*, que ens ofereix un treball més agradable. El podem descarregar des de http://sqliteadmin.orbmu2k.de/.

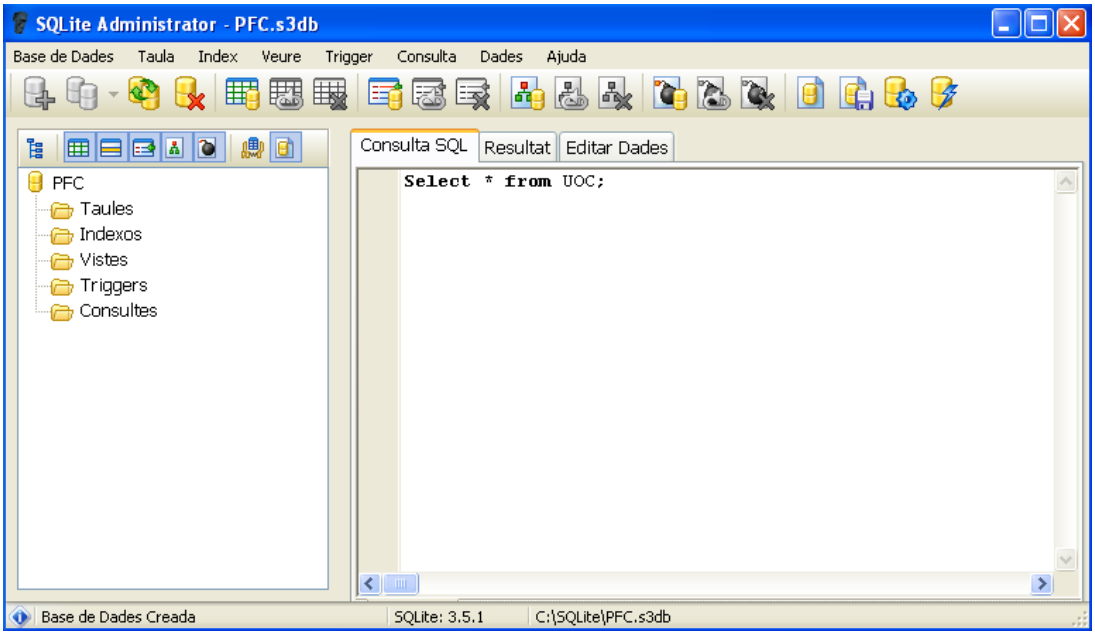

Figura 8 (SQLite Administrator)

4. Zona d'interès a treballar

Ens cal seleccionar un POI per tal de desenvolupar la nostra demo. Com que no tenim cap restricció a l'hora de fer la tria, utilitzarem per a la selecció amb els criteris de proximitat, volum d'informació i preferències personals. El de proximitat és molt important, perquè desenvolupar una aplicació que necessita de localització, el que vol dir que cal fer proves acurades i que les haurem de realitzar a l'aire lliure, just davant de l'indret amb el que treballarem, i ens cal minimitzar els desplaçaments.

4.1. Selecció

En el moment de l'elaboració d'aquest projecte la nostra residencia habitual està ubicada a l'Alt Empordà, a Garriguella concretament, però el nostre lloc de treball és a Girona. Estudiarem els POI's propers a aquests dos punts.

Propers a Garriguella trobem punts molt interessants:

- **Església de Santa Eulàlia de Noves (fotografia 1):** Església que data del segle XVIII i i que conté elements d'un antic temple preromànic del segle XI. Situada a Garriguella.

Fotografia 1 (Església de Santa Eulàlia de Noves)

- **Sant Pere de Rodes (fotografia 2):** monestir benedictí situat a Port de la Selva, data del segle X.

Fotografia 2 (Sant Pere de Rodes)

- **Sant Quirze de Colera:** abadia benedictina que data del segle IX situada a Rabós.

Fotografia 3 (Sant Quirze de Colera)

- **Ciutadella de Roses (fotografia 4):** fortificació renaixentista, situada a Roses, construïda al segle XV.

Fotografia 4 (Ciutadella de Roses)

- **La ciutadella d'Empúries (fotografia 5)**: ciutat Grega i Romana que data del segle V abans de Crist.

Fotografia 5 (La ciutadella d'Empúries)

- **El castell de Sant Ferran (fotografies 6 i 7):** fortalesa militar, construïda al segle XVIII, situat a Figueres. És el monument de majors dimensions de Catalunya.

Fotografia 6 (Entrada al Castell de Sant Ferran)

Fotografia 7 (Imatge aèria del Castell de Sant Ferran)

A la ciutat de Girona i les seves rodalies hi ha una gran quantitat de monuments / llocs d'interès.

- **Catedral de Santa Maria (fotografia 8):** catedral situada a la ciutat de Girona, la seva construcció es va dur entre els segles XIV i XVIII. Te la nau gòtica més ample del món.

Fotografia 8 (Catedral de Santa Maria de Girona)

- **Església de Sant Felix (fotografia 9):** basílica romànica (en la seva majoria) la construcció de la qual es va dur entre els segles XII i XVII.

Fotografia 9 (Església de Santa Felix)

- **Monestir de Sant Pere de Galligants (fotografia 10):** abadia benedictina d'estil romànic datada del segle X.

Fotografia 10 (Monestir de Sant Pere de Galligants)

- **El call Jueu (fotografia 11):** antigament el barri jueu de la ciutat, un del més ben conservat d'Espanya.

Fotografia 11 (El Call Jueu)

Podríem continuar mostrant possibles candidates durant molta estona gràcies a la riquesa cultural d'aquesta zona, però ja tenim prou candidates per el nostre treball. Per proximitat la guanyadora és la Església de **Santa Eulàlia de Noves de Garriguella**, i per preferències personals la **Catedral de Girona**. Amb la primera faré les proves de l'aplicació, i la segona serà la que farem servir per mostrar el funcionament.

4.2. Recerca d'imatges i informació

En el moment que volem cercar informació històrica dels monuments quan ens trobem el primer problema. Informació en trobem, però és molt més complicat trobar imatges (dibuixos, litografies, etc.) dels POI en les diferents èpoques històriques.

Per tant hem optat per un petit canvi d'estratègia per poder mostrar imatges en diferents moments del temps. El que farem és buscar alguna activitat que es faci de forma periòdica i que tingui un gran arxiu fotogràfic. Si parlem de Girona, aquesta elecció és ben senzilla, ja que cada any es duu a terme la mostra "Girona, Temps de Flors". Es tracta d'una exposició durant la qual molts dels espais importants de la ciutat com carrers, casses, patis de la ciutat (sobretot del Barri Vell) són guarnits amb motius florals. Les imatges següents en són un exemple:

Temps de flors 2012

Fotografia 12 (La catedral de Girona durant temps de flors de l'any 2012)

Temps de flors 2011

Fotografia 13 (La catedral de Girona durant temps de flors de l'any 2011)

Temps de flors 2010

Fotografia 14 (La catedral de Girona durant temps de flors de l'any 2010)

Temps de flors 2009

Fotografia 15 (La catedral de Girona durant temps de flors de l'any 2009)

Temps de flors 2008

Fotografia 16 (La catedral de Girona durant temps de flors de l'any 2008)

5. Aplicació pràctica

En arribar a aquest punt, hem aprofundit en les tecnologies que emprarem i ja tenim tot l'entorn de programació preparat. Passarem a posar en pràctica els coneixements adquirits, començant per una petita fase de disseny.

5.1. Disseny de l'aplicació

Recordarem que l'objectiu final d'aquest PFC és desenvolupar una petita aplicació demo per la visualització en 4D, tot i això plantejarem la solució de forma que pugui evolucionar cap a una versió més complerta. Hem d'obtenir una aplicació que permeti dur a terme visites en 4D d'un monument determinat (a partir d'ara els anomenarem POI de l'anglès *Point of Interest*), i ens basarem en el geoposicionament per identificar de quin POI es tracta i amb un framework d'AR generarem la visita.

a. Anàlisis de requeriments

L'usuari iniciarà una visita en 4D a partir de la seva localització: veurà en un plànol la seva posició i els POI propers (marcats amb una icona sobra el mapa), es permetrà fer zooms i desplaçar el mapa. Tot seguit l'usuari seleccionarà el POI a visitar pitjant amb el dit sobre la icona, a continuació podrà seleccionar el moment històric i finalment s'iniciarà la visita. El procés finalitzarà amb una acció manual de l'usuari.

Una anàlisi superficial de la solució ens porta a pensar que és una aplicació amb un sol perfil d'ús: l'usuari final. Això no és cert, tot i que no ho tractarem en profunditat (bàsicament només serà esmentat) existeix el perfil de l'administrador de POI, què és l'encarregat d'introduir i mantenir tota la informació relacionada amb els POI: administrar POI (afegir-ne de nous), administrar moments històrics (afegir-ne de nous i modificant els existents) i administrar les imatges (afegir-ne de noves i relacionar-les amb els POI's i els moments històrics, modificant-les). Repetim que aquesta part queda

fora de l'abast d'aquest PFC, però en un desenvolupament posterior s'hauria de proveir d'eines per dur a terme aquestes tasques.

A la figura 9 tenim els casos d'ús definits per un usuari tipus i el rol d'administrador.

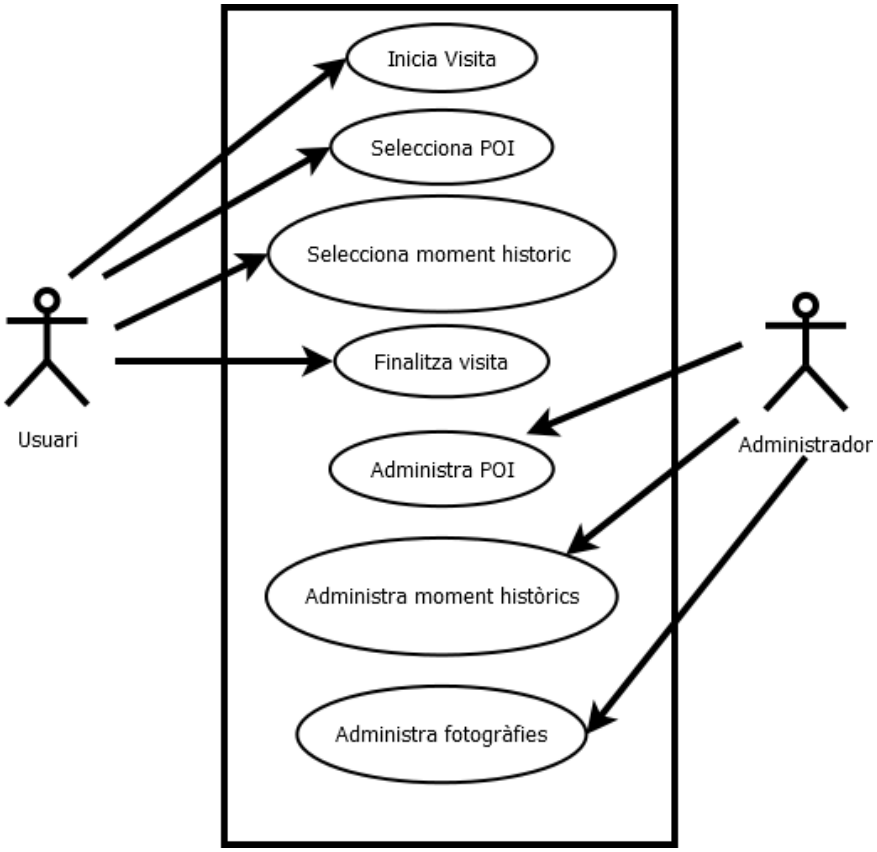

Figura 9 (Casos d'ús)

A continuació mostrarem el diagrama de seqüencia per un usuari i les accions que generen entre els diferents components de l'aplicació, quan es fa una visita 4D. (Figura 10)

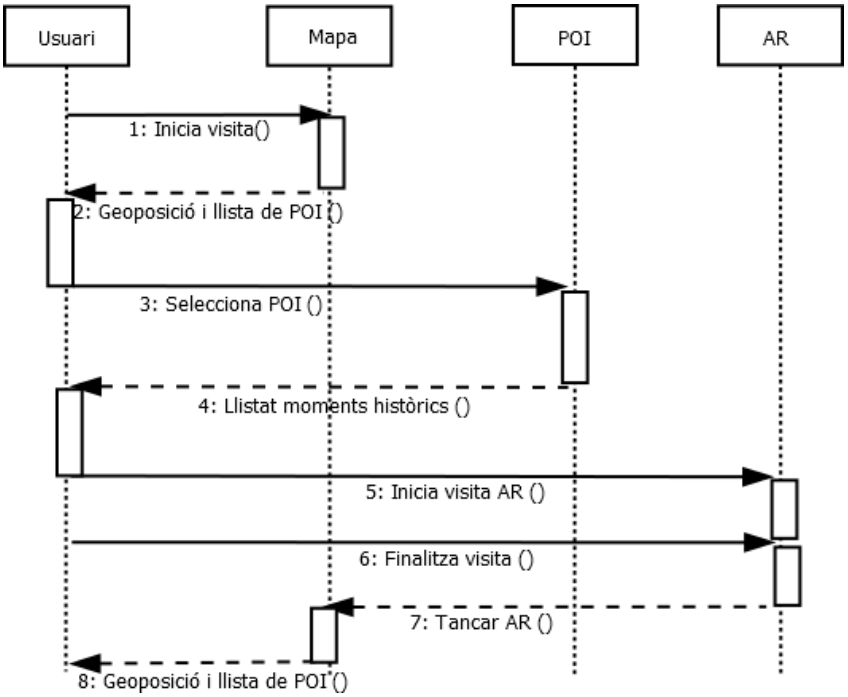

Figura 10 (Diagrama de seqüencia)

S'han pres tot un seguit de decisions per definir el que s'anomenen les "regles de negoci", tot aquell conjunt de normes que establirem nosaltres mateixos per definir el comportament tal i com pensem que ha de ser.

- No hi haurà gestió d'usuaris. Els usuaris no s'hauran de donar d'altra enlloc, ni es demanarà un login per accedir a l'aplicació.
- El número de POI's pot ser il·limitat, tants com vulgui l'administrador.
- Cada POI s'identificarà de forma inequívoca.
- Cada moment històric és inequívoc.
- La quantitat de moments històrics assignats a cada POI és il·limitada, tants com vulgui l'administrador.
- Un moment històric pot aparèixer a més d'un POI.
- Per a un POI i un moment històric hi pot haver una quantitat il·limitada d'imatges però cadascuna d'elles només fa referència a un únic punt en el mapa que ha de ser inequívoc.

A la figura 11 veiem el diagrama d'entitat relació que s'aplicaria per el nostre cas.

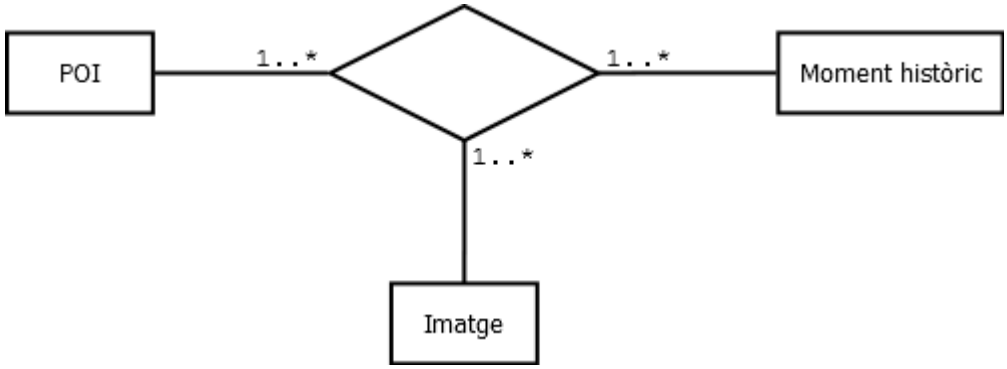

Figura 11 (Diagrama ER)

b. Anàlisis de requisits

A l'enunciat l'únic requisit al qual es feia referència és sobre el sistema operatiu: ha de ser Android. Nosaltres definirem els següents:

- Versió mínima d'Android 2.2: el nostre PFC funcionarà sobre la versió 2.2 (*Froyo*) i posteriors. La raó d'aquesta decisió és la gran quantitat de terminals que tenen aquesta versió.
- No hi ha requisits de seguretat: No encriptarem ni protegirem cap dada.
- Emmagatzematge d'imatges: Es guardaran a la base de dades, no seran accessibles des d'un explorador d'arxius.
- La informació generada per nosaltres s'emmagatzemarà en local: La base de dades amb la informació dels POI i les seves imatges residirà en el dispositiu. No hi haurà peticions a la xarxa per recuperació de dades, a excepció de les necessitats del component de visualització de mapes (però són necessitats definides per el proveïdor).
- Serveis actius per geoposicionament: serà requisit obligatori per iniciar una visita tenir un servei de GPS i 3G o Wifi actius. No desenvoluparem avisos ni ajudes per la seva connexió.

El diagrama de classes que obtenim el veiem tot seguit, a la figura 12. Seran les nostres classes bàsiques i la seva relació.

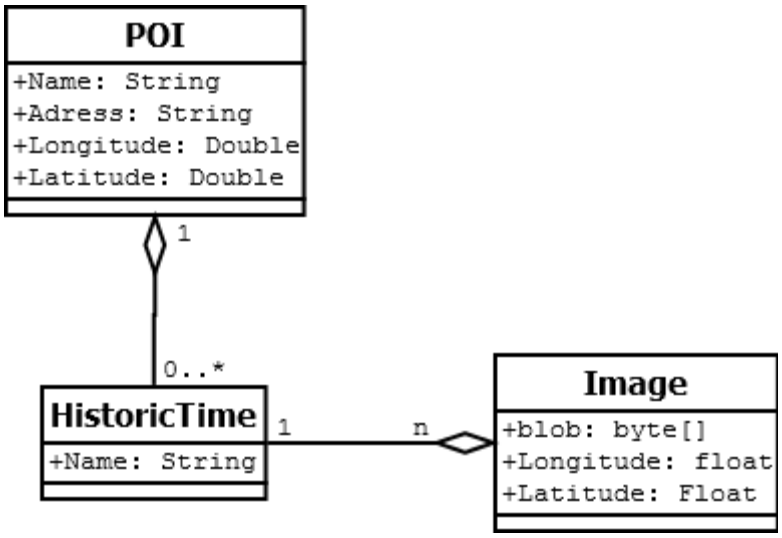

Figura 12 (Diagrama de classes)

c. Proves

El nostre procés de desenvolupament ens donarà com a fruit la demo desitjada, però per tal de valorar la seva qualitat tindrem en compte els següents jocs de proves.

- Geoposicionament amb GPS.
- Geoposicionament amb 3G.
- Visita a un POI amb un únic moment històric.
- Visita a un POI amb més d'un moment històric.

d. Model de treball

Com a model de treball seguirem un model en cascada. Aquest model incorpora les fases que llistarem a continuació:

- Anàlisis de requisits.
- Disseny de l'aplicació.
- Codificació.
- Proves.

- Implantació.
- Manteniment.

En el nostre cas les dues últimes fases són pràcticament inexistents.

5.2. Disseny de la base dades

A l'apartat 3.4 Base de dades es va parlar del sistema gestor que es faria servir (SQLite) i de com emmagatzemaríem les dades en el propi dispositiu. En l'apartat anterior hem vist les entitats amb les quals treballarem, a partir d'aquestes dissenyarem la nostra base de dades (Figura 13).

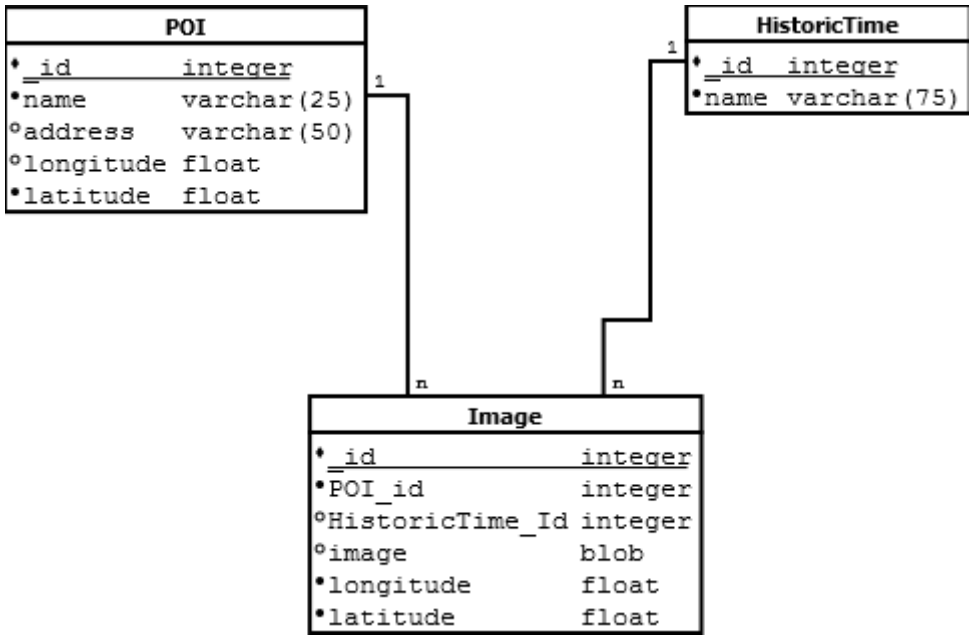

Figura 13 (Base de dades)

Tenim 3 úniques taules.

1. *POI*: emmagatzemarem les dades dels nostres POI's. Les coordenades seran guardades en format decimal (el seu ús en el codi Java ens serà molt més senzill que la notació sexagesimal), dividides en dos camps diferes: longitud i latitud.
2. *HistoricTime*: es guardaren les dades dels diferents moments històrics.
3. *Image*: Per cada POI i un moment històric emmagatzemarem les seves imatges. Cada imatge va localitzada amb la seva longitud i latitud.

Android per treballar amb una base de dades SQLite, té dos requisits:

1. Totes les bases de dades han de tenir la taula **android_metadata**. Es pot crear amb el següent script.

```
CREATE TABLE "android_metadata" ("locale" TEXT DEFAULT
'en_US')
```

On cal inserir el següent registre:

```
INSERT INTO "android_metadata" VALUES ('en_US')
```

2. Totes les taules han de tenir un camp anomenat **_id** de tipus **integer** com a clau primària. Això no ens ha comportat gaires problemes en el disseny, bàsicament l'únic canvi per adaptar-nos als requisits ha estat un canvi de nom.

   Presentem les sentencies SQL per la creació de les taules.

   - Taula *POI*

```
CREATE TABLE [POI] (
      [_id] INTEGER  NOT NULL PRIMARY KEY,
      [Name] NVARCHAR(25)  NULL,
      [Adress] NVARCHAR(50)  NULL,
      [Longitude] FLOAT  NULL,
      [Latitude] FLOAT  NULL
)
```

   - Taula *HistoricTime*

```
CREATE TABLE [HistoricTime] (
      [_id] INTEGER  NOT NULL PRIMARY KEY,
      [Name] NVARCHAR(25)  NULL
)
```

- Taula *Image*

```
CREATE TABLE [Image] (
      [_id] INTEGER  NOT NULL PRIMARY KEY,
      [POI_Id] integer,
      [HistoricTime_id]  integer,
      [Image] blob,
      [Longitude] float,
      [Latitude] float,

       FOREIGN KEY (POI_Id) REFERENCES POI(_id),
       FOREIGN KEY (HistoricTime_Id) REFERENCES
            HistoricTime(_id)
       UNIQUE (POI_id, HistoricTime_id, Longitude, Latitude)
)
```

5.3. Desenvolupament de la visualització 4D

En aquest apartat es detalla pas a pas el procés de codificació de la solució. Hem dividit aquesta tasca en 4 subtasques:

1. Geoposicionament: on hem de ser capaços de localitzar la nostre posició en un mapa i representar-la. Més tard, sobre aquest mateix mapa presentarem els POI que tinguem definits propers a la nostra posició.
2. Accés a la base de dades: Els POI's i la informació relativa d'aquests estan emmagatzemats en una base de dades SQLite, hem d'atacar-la per extreure'n la informació. Un punt a tenir en compte, és la distribució d'aquesta base de dades conjuntament amb la nostra aplicació.
3. Representació de POI: amb la informació recuperada de la nostre base de dades representarem el POI sobre el mapa.
4. Aplicació de framework de realitat augmentada: Amb l'ajuda de la càmera del nostre dispositiu mostrarem l'escena real amb les imatges superposades d'altres èpoques de la història.

No entrarem en detall a les tasques bàsiques de programació en Android, com poden ser botons, ens centrarem en les parts importants de l'aplicació de SIG.

5.3.1. Geoposicionament sobre el mapa

Android i el seu fabricant: Google, ofereixen el control *MapView* (Figura 14) per a la representació de mapes en les nostres aplicacions. L'incrustarem en el nostre programari per tal de mostrar la zona on estem situats i poder localitzar visualment els POI.

(Figura 14 MapView)

L'ús del component MapView, ens obliga a definir l'objectiu de compilació (*Build target*) com a **Google APIs (Platform 2.2)** (Figura 15). Normalment faríem servir Android 2.2, però l'API de Google Maps no forma part de la distribució normal d'Android.

Project Build Target

Target Name	Vendor	Platform	AP...
Android 1.5	Android Open Source Project	1.5	3
Google APIs	Google Inc.	1.5	3
Android 1.6	Android Open Source Project	1.6	4
Google APIs	Google Inc.	1.6	4
Android 2.1	Android Open Source Project	2.1	7
Google APIs	Google Inc.	2.1	7
Android 2.2	Android Open Source Project	2.2	8
Google APIs	Google Inc.	2.2	8
GALAXY Tab Addon	Samsung Electronics Co., Ltd.	2.2	8
GALAXY Tab Addon	Samsung Electronics Co., Ltd.	2.2	8
Android 2.3.1	Android Open Source Project	2.3.1	9
Google APIs	Google Inc.	2.3.1	9
Android 2.3.3	Android Open Source Project	2.3.3	10
Google APIs	Google Inc.	2.3.3	10
Android 3.0	Android Open Source Project	3.0	11
Google APIs	Google Inc.	3.0	11
Android 3.1	Android Open Source Project	3.1	12
Google APIs	Google Inc.	3.1	12
Android 3.2	Android Open Source Project	3.2	13
Google APIs	Google Inc.	3.2	13
Android 4.0	Android Open Source Project	4.0	14
Google APIs	Google Inc.	4.0	14
Android 4.0.3	Android Open Source Project	4.0.3	15
Google APIs	Google Inc.	4.0.3	15

Figura 15 (Selecció del l'objectiu de compilació)

En el cas que treballem amb l'emulador, s'ha de crear un AVD on l'objectiu de compilació sigui **Google APIs (Google Inc.) – API Level 8**. Com es veu a la següent figura (16).

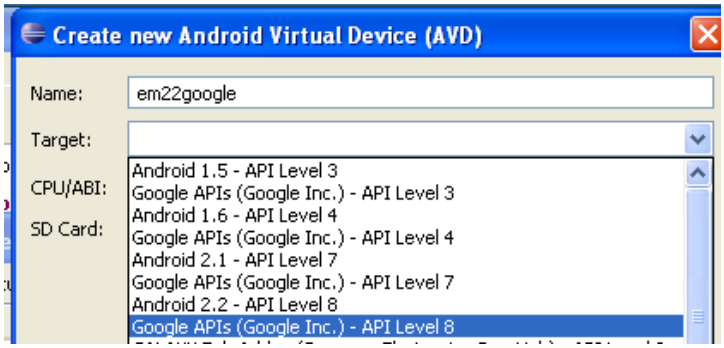

Figura 16 (Creació d'una AVD per emulació amb *MapView*)

Per què un mapa amb aquesta tecnologia sigui funcional, Google ens obliga a obtenir el seu permís per fer servir el component: al directori on tenim instal·lat el Java JDK trobem l'eina **keytool**, que ens ajudarà a generar una clau a partir de la informació de la nostra màquina, al directori *C:\Documents and Settings\<usuari>\.android* trobarem l'arxiu **debug.keystore**, que és l'arxiu a partir del qual es crearà la clau. Executant la instrucció següent a la línia de comandes:

```
<Directori Java>\bin>keytool -list -alias androiddebugkey -keystore
"<C:\Documents and Settings\<usuari>\.android\debug.keystore"
-storepass android -keypass android
```

Obtindrem una clau amb un format semblant al que presentem a continuació:

```
Huella digital de certificado (MD5):
14:98:EA:56:A4:47:55:9B:97:EA:0A:9D:44:CF:8C:E0
```

Un cop generada la clau, podem anar a la pàgina https://developers.google.com/android/maps-api-signup?hl=es i generar la clau per poder treballar amb l'API de Google Maps a les nostres aplicacions. La figura 17 és un exemple de com introduir les dades obtingudes en els passos anteriors.

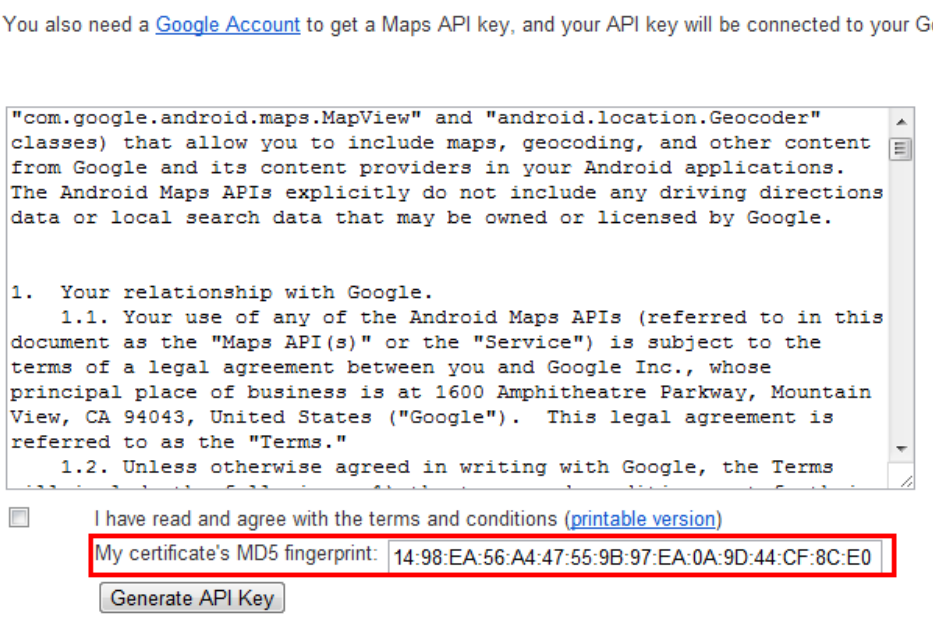

Figura 17 (Obtenció de la clau per l'API de Google Maps)

Google ens generarà una *apiKey*, que hem d'escriure al nostre codi, concretament en el *layout* on es trobi el component *MapView*. En el nostre cas, hem creat el *layout* **maplayout** i s'ha incrustat un *MapView*, en les propietats d'aquest hem inserit la nostra clau.

```
<com.google.android.maps.MapView
    android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="434dp"
    android:apiKey="apiKEY"
    android:clickable="true" >
</com.google.android.maps.MapView>
```

El pas següent, és dotar al nostre mapa amb la capacitat d'indicar la nostra posició actual. Crearem la classe **Visor4dmap** que estén la classe *MapActivity;* els controls *MapView* només poden ser inclosos en classes que heretin de d'aquesta classe. La classe *MapActivity* és l'encarregada de controlar els fils d'execució en segon pla, la connexió a internet per aconseguir dades, gestionar la memòria cau, controlar les animacions, etc.

```
public class Visor4dmap extends MapActivity {
	private MapView map;
	private MapController controller;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.maplayout);

		initMapView();
		initMyLocation();
	}

	private void initMapView() {
		map = (MapView) findViewById(R.id.map);
		controller = map.getController();
		map.setSatellite(true);
		map.setBuiltInZoomControls(true);
	}

	private void initMyLocation() {
		final MyLocationOverlay overlay = new
			MyLocationOverlay(this, map);

		overlay.enableMyLocation();
		overlay.enableCompass();
		overlay.runOnFirstFix(new Runnable() {

		public void run() {
			controller.setZoom(19);
			controller.animateTo(overlay.getMyLocation());
		}
		});
		map.getOverlays().add(overlay);
	   }

	@Override
	protected boolean isRouteDisplayed() {
		return false;
	}
}
```

És important l'objecte *MapController* que ens servirà per posicionar-nos, fer zooms i desplaçar els mapes. L'objecte *map* el fem servir per canviar a vista de satèl·lit i per mostrar / amagar els controls de zoom.

El mètode *initMyLocation* serà l'encarregat de marcar la nostra posició actual i ho farà gràcies a que s'ha fet servir la classe *MyLocationOverlay*. Un *overlay* és una capa que es superposa al mapa, i en aquest cas és una capa que mostra la nostra posició actual. Els primers passos són habilitar les actualitzacions de posició (*enableMyLocation*) i les de orientació (*enableCompass*). El mètode *runOnFirstFix* indica el comportament del primer cop que la capa rep una nova lectura de posició, en

aquest cas, activem el zoom i movem el mapa cap a la situació actual. La classe *MyLocationOverlay* implementa un *LocationListener* (més endavant hi entrarem en més detall), això vol dir que està pendent de les actualitzacions de posició cada cert temps, amb el mètode *setLocationUpdateMinTime* es pot indicar aquest temps, per defecte el seu valor és 0 i és com el deixarem nosaltres.

Per últim, cal modificar el fitxer **AndroidManifest** del nostre projecte. Cal definir els permisos per els serveis que necessitarem, en aquest cas localització i internet.

```
<!—Permissos per obtenir la geoposició -->
<uses-permission
       android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
       android:name="android.permission.ACCESS_FINE_LOCATION" />

<!—Permissos per accedir a la xarxa-->
<uses-permission
       android:name="android.permission.INTERNET" />
```

I les llibreries necessàries:

```
<uses-library android:name="com.google.android.maps" />
```

Més endavant retornarem al component *MapView* i al geoposicionament, ja que ens falta indicar la posició dels nostres POI's. Ho veurem un cop hàgim treballat l'accés a la base de dades per recuperar les localitzacions.

5.3.2. Accés a la base de dades

Per recuperar les dades de la nostra base de dades, disposem de la classe *SQLiteOpenHelper*. Per tant, estendrem aquesta classe en la nova classe **Visor4dDataBase**.

```
public class Visor4dDataBase extends SQLiteOpenHelper {
	private static String DB_PATH =
			"/data/data/uoc.PFC.sig/databases/";
	private static String DB_NAME = "pfc.db";

	private SQLiteDatabase myDataBase;
	private final Context myContext;

	public Visor4dDataBase (Context context) {
		super(context, DB_NAME, null, 1);
		this.myContext = context;
	}

	public void createDataBase() throws IOException {
		boolean dbExist = checkDataBase();

		if (dbExist) {
		}
		else {
			this.getReadableDatabase();

			try {
				copyDataBase();
			}
			catch (IOException e)
			{
				throw new Error("Error copying database");
			}
		}
	}

	private boolean checkDataBase() {

		SQLiteDatabase checkDB = null;

		try {
			String myPath = DB_PATH + DB_NAME;
			checkDB =
				SQLiteDatabase.openDatabase(myPath, null,
					SQLiteDatabase.OPEN_READONLY);
		}
		catch (SQLiteException e)
		{
			// database does't exist yet.
		}

		if (checkDB != null) {
			checkDB.close();
		}

		return checkDB != null ? true : false;
	}
```

```
	private void copyDataBase() throws IOException {

		InputStream myInput =
			myContext.getAssets().open(DB_NAME);
		String outFileName = DB_PATH + DB_NAME;

		OutputStream myOutput = new
			FileOutputStream(outFileName);

		byte[] buffer = new byte[1024];
		int length;
		while ((length = myInput.read(buffer)) > 0) {
			myOutput.write(buffer, 0, length);
		}

		myOutput.flush();
		myOutput.close();
		myInput.close();
	}

	public void openDataBase() throws SQLException {
		String myPath = DB_PATH + DB_NAME;
		myDataBase = SQLiteDatabase.openDatabase(myPath, null,
				SQLiteDatabase.OPEN_READWRITE);
	}

	@Override
	public synchronized void close() {
		if (myDataBase != null)
			myDataBase.close();

		super.close();
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
	}

	@Override
	public void onUpgrade(SQLiteDatabase db,
		int oldVersion, int newVersion) {
	}
}
```

Les aplicacions Android que treballen amb SQLite, emmagatzemen per defecte el fitxer de la base de dades a */data/data/<el nom del paquet>/databases/*, això és molt important per saber com accedirem i distribuirem les nostres dades. Tenim tres opcions

1. Generar els scripts de creació de la base de dades, i al nostre objecte **Visor4dDataBase** al mètode *OnCreate* executar-los. Aquesta opció permet una distribució molt senzilla del fitxer (el generem en *runtime*) però no

permet una bona gestió de les dades, ja que necessitem de mètodes en el nostre programari per inserir / modificar.

2. Generar la base de dades i copiar-la al directori de treball. Un cop generada la guardarem al nostre directori *assets* del projecte, i d'allà la distribuirem. Aquesta opció permet no haver de fer codi per la gestió de les dades, però complica el manteniment de les dades, qualsevol actualització implica tornar a copiar la BD al nostre dispositiu.
3. Generar la base de dades i copiar-la a l'emmagatzemament extern del dispositiu (targeta SD). És una opció molt semblant a l'anterior, però estem afegint requisits al dispositiu, ja que ha de disposar d'una memòria externa.

Hem triat la segona opció, ja que l'abast del projecte no està contemplada la gestió de la BD simplement la seva consulta, també per la flexibilitat que ofereix el poder gestionar les dades a partir d'un administrador com *SQLite Administrator* i per no necessitar recursos externs.

Retornem a la nostra classe **Visor4dDataBase** i el seu tractament. Al mètode *createDataBase* verifiquem si el dispositiu ja conté la base de dades, en cas negatiu la copiarem del nostre directori *assets* cap a */data/data/< el nom del paquet >/databases/* (*copyDataBase*). Finalment el mètode *openDataBase* proveeix la possibilitat per accedir a les dades. Així doncs l'accés el farem com ve a continuació:

```
private Visor4dDataBase myDataBase;

myDataBase = new Visor4dDataBase (this);

try {
      myDataBase.createDataBase();
} catch (IOException e) {
      e.printStackTrace();
}

try{
      myDataBase.openDataBase();

      // TO DO - TRACTAMENT DE DADES

}finally{
      myDataBase.close();
}
```

No hem explicat encara com accedir a les taules en concret, resulta molt senzill des de la nostra classe **Visor4dDataBase** executarem una consulta SQL, i posteriorment recorrerem el cursor per obtenir les dades.

```
Cursor c = myDataBase.rawQuery(("SELECT Name, adress, longitude,
      latitude FROM POI"), null);
```

Volem que la nostra classe **Visor4dDataBase** encapsuli totes les accions sobre la BD. Per tant s'han de crear els mètodes que retornin llistes de dades per poder ser tractades per altres parts dels projecte, neix la necessitat de crear les nostre classes.

5.3.2.1. Creació de classes bàsiques

Al capítol 5.1 Disseny de l'aplicació ja hem vist la necessitat i el disseny que han de tenir les nostres classes, ara les implementarem en codi Java.

```
public class POI {
      int id;
      String name;
      String adress;
      Double longitude;
      Double latitude;
      List<HistoricTime> times;

      public POI() {
            this.name = "";
            this.adress = "";
            this.longitude = 0.0;
            this.latitude = 0.0;
      }
}
```

```
public class HistoricTime {
      String name;
      List<Image> images;

      public HistoricTime () {
            this.name = "";
      }
}
```

```
public class Image {
      float longitude;
      float latitude;
      byte[] blob;

      public Image() {
      }
}
```

5.3.2.2. Obtenció de les dades dels POI

Ara que ja tenim les nostres classes codificades podem passar a obtenir els valors de les nostres taules i fer servir aquestes dades de forma transparent a la resta del projecte. Al nostra classe **Visor4dDataBase** hi afegirem els següents mètodes.

```
public List<POI> ListPOI()
{
      List<POI> myPOIs = new ArrayList<POI>();

      Cursor c = myDataBase.rawQuery(("SELECT Name, adress,
                              longitude, latitude, _id FROM POI"),
                        null);
      if (c.moveToFirst()) {
            do {
                  POI auxPOI = new POI();
                  auxPOI.name = c.getString(0);
                  auxPOI.adress = c.getString(1);
                  auxPOI.longitude = c.getDouble(2);
                  auxPOI.latitude = c.getDouble(3);
                  auxPOI.id = c.getInt(4);

                  myPOIs.add(auxPOI);

                  } while(c.moveToNext());
            }

            c.close();

            return myPOIs;
   }
```

El mètode anterior és un simple recorregut de la taula POI per tal d'obtenir una llista que és la que retornarem a l'objecte que faci la crida.

Donat un POI i els moment històric a visitar, obtindrem les imatges.

```
public List<Image> ListImage(int poi_id, int historictime_id)
{
      List<Image> images = new ArrayList<Image>();

      Cursor c = myDataBase.
            rawQuery((" SELECT Image, longitude, latitude " +
                  " FROM Image IMG " +
                  " WHERE IMG.POI_id = " +
                        String.valueOf(poi_id) +
                  " AND IMG.HistoricTime_id = "
                    + String.valueOf(historictime_id)), null);

            if (c.moveToFirst()) {
                  do {
                        Image img;
                        img = new Image();
                        img.blob =  c.getBlob(0);
                        img.longitude = c.getFloat(1);
                        img.latitude = c.getFloat(2);
                        images.add(img);
                  } while(c.moveToNext());
            }

            c.close();

            return images;
}
```

I finalment donat un POI obtindrem una llista amb la seva descripció dels moments històrics relacionats.

```
public LinkedList<HistoricTime> ListHistoricTime(int POI_id)
{
      LinkedList<HistoricTime> temps = new
                              LinkedList<HistoricTime>();

      Cursor c = myDataBase.rawQuery((" SELECT DISTINCT TIM._Id, " +
                                    " TIM.name " +
                                    " FROM HistoricTime TIM" +
                                    " JOIN Image IMG ON TIM._id =
                                        IMG.HistoricTime_Id" +
                                     " WHERE IMG.POI_id =" +
                              String.valueOf(POI_id)), null);

            if (c.moveToFirst()) {
                  do {
                        HistoricTime auxTime;
                        auxTime = new HistoricTime();

                        auxTime.id = c.getInt(0);
                        auxTime.name = c.getString(1);

                        temps.add(auxTime);
                  } while(c.moveToNext());
            }

      c.close();
      return temps;
}
```

5.3.2.3. Càrrega de dades

Com s'ha comentat amb anterioritat, no preveiem la possibilitat de manipular les dades des de la nostre demo, per tant, la base de dades ja haurà de contenir de de l'inici les dades necessàries per el funcionament. Adjuntem els scripts necessaris per generar l'estructura de taules.

- Inserim llista de POI

```
Insert into POI(name, adress, Longitude, Latitude) Values ("Catedral
de Girona","Plaza de la Catedral, 6, Girona", 2.826449, 41.987516)

Insert into POI(name, adress, Longitude, Latitude)
Values ("Banys arabs","Carrer Francesc Samsó", 2.825676,41.988321)

Insert into POI(name, adress, Longitude, Latitude)
Values ("Esglesia de Sant Felix", "Pujada de Sant Feliu,
29",2.82449, 41.9875799)

Insert into POI(name, adress, Longitude, Latitude)
Values ("Esglesia Garriguella","Plaça de l'esglesia", 3.064982,
42.342125)
```

- Inserim llista de HistoricTime

```
Insert into HistoricTime(Name) Values('Període actual')

Insert into HistoricTime(Name) Values('Temps de Flors 2012')

Insert into HistoricTime(Name) Values('Temps de Flors 2011')

Insert into HistoricTime(Name) Values('Temps de Flors 2010')

Insert into HistoricTime(Name) Values('Temps de Flors 2009')

Insert into HistoricTime(Name) Values('Temps de Flors 2008')
```

- Inserim a Image la relació entre el POI i HistoricTime.

```
Insert into image (POI_id, HistoricTime_id, Longitude, latitude)
values(4,7,3.064597,42.342294)

Insert into image (POI_id, HistoricTime_id, Longitude, latitude)
values(1,8,2.826449, 41.987516)

Insert into image (POI_id, HistoricTime_id, Longitude, latitude)
values(1,9,2.826449, 41.987516)

Insert into image (POI_id, HistoricTime_id, Longitude, latitude)
values(1,10,2.826449, 41.987516)

Insert into image (POI_id, HistoricTime_id, Longitude, latitude)
values(1,11,2.826449, 41.987516)

Insert into image (POI_id, HistoricTime_id, Longitude, latitude)
values(1,12,2.826449, 41.987516)
```

La inserció d'un camp blob en una taula SQLite mitjançant sentencies SQL no és pas trivial. L'eina *SQLite Administrator* que es va triar a l'apartat 3.4 Base de dades tampoc ens és d'utilitat. Hem cercat una alternativa senzilla, i hem trobat el *SQLite Manager* (figura 18), que és un complement per al conegut navegador *Mozilla Firefox* que permet la gestió de bases de dades. Aquesta eina permet inserir imatges a un camp BLOB molt fàcilment.

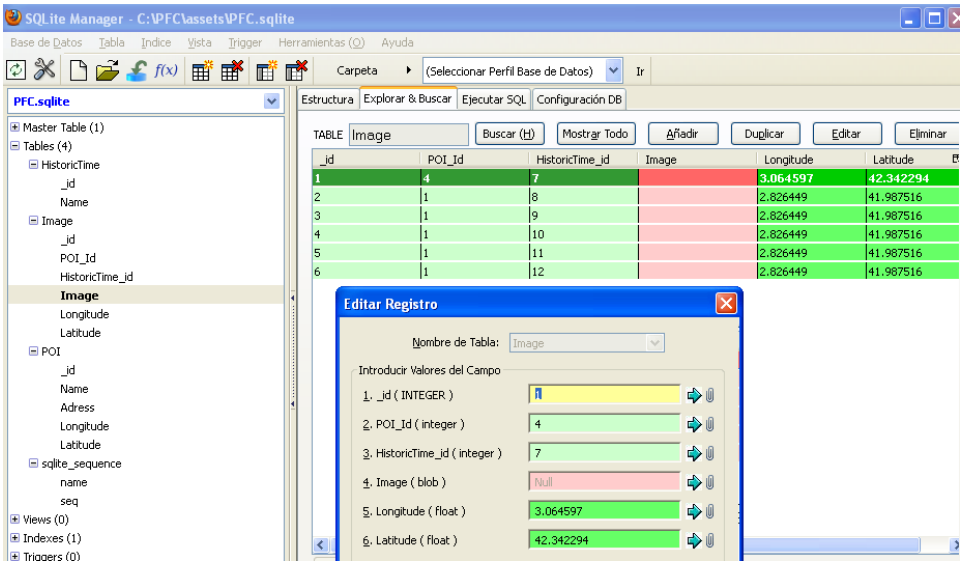

Figura 18 (SQLite Manager)

5.3.3. Visualització de POI en el mapa

Un cop som capaços de senyalar la nostre posició en el mapa, i d'obtenir la localització dels nostres POI emmagatzemats a la BD (com hem fet en capítols anterior), el pas més natural a seguir és proveir el mapa amb la capacitat de visualitzat els POI. Aprofitarem també per donar indicacions quan estem tant a prop d'un POI com per iniciar la visita.

Un POI el senyalarem amb la icona clàssica de Google maps (Figura 19a) , i quan estem a menys de 50 metres, canviarem el color d'aquest icona per el color blau (Figura 19b). Aquestes imatges les hem de desar al directori */res/drawable* del nostre projecte, així les imatges estaran disponibles per qualsevol mida de pantalla.

Figura 19a (POI)

Figura 19b (POI proper per visitar)

Un altre cop, i com ja hem fet anteriorment per visualitzar la nostre posició, hem de treballar amb capes (*overlays*) per indicar la posició dels POIs. Aquest cop però, no podem fer servir la capa que ens ofereix Google i ens cal crear la nostra pròpia capa: li direm **Visor4dmaplay**, i estendrà la classe *Overlay*.

No desitgem que sigui la capa la que accedeixi a la base de dades ja que incrementaria el número de consultes, a la classe **Visor4dmap** crearem els mètodes, *initDataBase* per obrir la connexió, i *GetListPOI* que es connecta a la BD per obtenir la llista de punts i els hi passarem a la capa.

```
public class Visor4dmap extends MapActivity {
      private static Context context;
      List<POI> POIs;
      Location MyLocation;

      public void GetListPOI(){
            dades.openDataBase();
            try{
                  POIs = new ArrayList<POI>();
                  POIs = dades.ListPOI();
            } finally {
                  dades.close();
            }
      }
```

```
public void initDataBase()
{
      dades = new DataBaseHelper(this);

      try {
            dades.createDataBase();
      } catch (IOException e) {
            e.printStackTrace();
      }
}
```

La capa **Visor4dmaplay** té el mètode *draw*, on recorrem la llista de POIs i per cada un d'ells es valora la distancia a la posició *MyLocation*. El mètode *IsNear* calcula si la distancia entre dos punts és menor a la distancia determinada (l'hem prefixada en 50 metres).

```
public class Visor4dmaplay extends Overlay  {
      private static Context context;
      List<POI> POIs;
      Location MyLocation;

      public boolean isNear(GeoPoint geoPoint, Location MyLocation,
            int Distance)
      {
            Location POIlocation = new Location("geopoint");
            POIlocation.setLatitude(geoPoint.getLatitudeE6() / 1e6);
            POIlocation.setLongitude(geoPoint.
                        getLongitudeE6() / 1e6);

            return (MyLocation.distanceTo(POIlocation) < Distance);
      }
```

El mètodes *IsNear* no és del tot acurat, degut a que només considera la distància en com la separació entre nosaltres i el POI, i no és el mateix apropar-nos al POI per la façana davantera que per la del darrera, perquè obtenim visions diferents del mateix POI. La forma de solucionar aquest problema seria conèixer l'orientació del POI i la posició de la que es va prendre la imatge.

```
	@Override
	public void draw(Canvas canvas, MapView mapView, boolean
		shadow)
	{
		Paint p = new Paint();

		Projection projection = mapView.getProjection();


		for(int i=0; i <= (POIs.size()-1); i++) {

		   int ilatitud = (int)
		      (POIs.get(i).Latitud.doubleValue()*1E6);
		   int ilongitud = (int)
		      (POIs.get(i).Longitud.doubleValue()*1E6);

		   GeoPoint geoPoint =  new GeoPoint(ilatitud
		                        ,ilongitud);

		   if (shadow == false) {

		      Point centro = new Point();
		      projection.toPixels(geoPoint, centro);

		      Bitmap bm;

		      if (MyLocation == null){
		            bm = BitmapFactory.decodeResource
		                  (mapView.getResources(),
		                  R.drawable.marcador_google_maps);

		            canvas.drawBitmap(bm,centro.x-bm.getWidth()
		            , centro.y - bm.getHeight(), p);
		      }
		      Else {

		            if (isNear(geoPoint, MyLocation, 50))
		                  bm = BitmapFactory.decodeResource
		                        (mapView.getResources(),
		                        R.drawable.marcador_google_maps2
		                        );
		            else
		                  bm = BitmapFactory.decodeResource
		                  (mapView.getResources(),
		                  R.drawable.marcador_google_maps);
		            }

		      canvas.drawBitmap(bm, centro.x - bm.getWidth(),
		            centro.y - bm.getHeight(), p);
		      }
		}

	public void setMyLocation(Location location){
		if (location != null)
		      MyLocation = location;
	}

}
```

El mètode *SetMyLocation* facilita una porta d'entrada perquè la capa pugui disposar de la localització actual, la feina d'obtenir la posició la deleguem al **Visor4dmap**.

5.3.4. Geoposicionament (II)

El que hem fet a l'apartat 5.3.1.1 Geoposicionament sobre el mapa ens ha servit per la visualització, però nosaltres necessitem obtenir les dades per poder calcular distancies. Cal que habilitem la possibilitat de rebre les actualitzacions de la posició, i això es fa mitjançant el mètode *requestLocationUpdates* de la classe *LocationManager*.

```
public class Visor4dmap extends MapActivity {
      private Location MyLocation;
      private LocationManager locManager;
      private LocationListener locListener;
      ……..


      private void getMyLocation()
      {
            locManager = (LocationManager)
                        getSystemService(this.LOCATION_SERVICE);

            Criteria criteria = new Criteria();
            String best = locManager.getBestProvider(criteria,
                                                      true);

            Location loc = locManager.getLastKnownLocation(best);

            locListener = new LocationListener() {
                  public void onLocationChanged(Location location) {
                        MyLocation = location;
                        om.setMyLocation(MyLocation);
            }

            public void onProviderDisabled(String provider){
                  Log.i("", "Provider OFF");
            }

            public void onProviderEnabled(String provider){
                  Log.i("", "Provider ON ");
            }

            public void onStatusChanged(String provider, int status,
                        Bundle extras){
                  Log.i("", "Provider Status: " + status);
            }
          }

          locManager.requestLocationUpdates(best, 3000,
                                    0.5f, locListener);
      }
}
```

Hem creat el mètode *getMyLocation* que mitjançant la classe *LocationManager* extreu informació de quins són els millors serveis de localització actual (*getBestProvider*) i amb el *requestLocationUpdates* habilitem que Android ens informi sobre els canvis de posició (ho hem configurat cada 3.000 mili-segons, i amb qualsevol canvi de posició superior a 0,5 metres). El mètode *getBestProvider* retorna el millor proveïdor tenint en compte restriccions: energia, precisió, etc. Hem decidit no aplicar cap restricció.

La classe *LocationListener* és l'encarregada de demanar periòdicament al sistema (com s'ha definit en el paràgraf anterior *requestLocationUpdates*) les actualitzacions de posició. És molt important el mètode *onLocationChanged* ja que és el que s'executarà cada cop que hi hagi un canvi de posició, és aquí on obtenim el valor de la nostra posició actual i li passarem a la capa (*om.setMyLocation(MyLocation)*).

El darrer pas, és configurar la nostra capa (**Visor4dmaplay**), i això ho farem modificant el mètode *onCreate* de la classe **Visor4dmap**, on cridarem els mètodes *SetPOI* i *SetMyLocation*.

```
public class Visor4dmap extends MapActivity {
	private Visor4dmaplay om;
	private Location MyLocation;
	……


	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.maplayout);

		initDataBase();
		initMapView();
		initMyLocation();
		getMyLocation();

		om = new Visor4dmaplay();
		List<Overlay> capas = map.getOverlays();
		om.setPOI(POIs);
		om.setMyLocation(MyLocation);

		capas.add(om);
		map.postInvalidate();

	}
}
```

A la següent imatge (Figura 20) mostrarem gràficament els canvis que hem anat aplicant al nostre mapa.

1. MapView bàsic

2. MapView amb capa de geoposició

3. MapView amb capa de geoposició i recuperació de POI de la BD.

4. MapView amb geoposicionament, recuperació de POI de la BD i càlcul de distancies (mètode *isNear*)

Figura 20 (Evolució del nostre mapa)

L'última de les accions que podrem fer sobre el mapa és començar una visita. S'ha fet de la manera més natural possible i que és donant un petit cop amb el dit a la localització. Aquest event es diu *onTap* i és part de la classe **Visor4dmaplay** (no del *mapview*). Des d'aquí anirem a un desplegable on es mostraran els moments històrics associats al POI.

```
public class Visor4dmaplay extends Overlay  {
      ……
      @Override
      public boolean onTap(GeoPoint point, MapView mapView)
      {
            if (MyLocation != null) {

                  for(int i=0; i <= (myPOIs.size()-1); i++) {

                        if (isNear(point, MyLocation, 10)) {
                              Context contexto =
                                    mapView.getContext();
                        Intent intent = new
                              Intent(mapView.getContext(),
                              Visor4dSpinner.class);

                              intent.putExtra("POI_Id",
                                    myPOIs.get(i).id);
                              contexto.startActivity(intent);
                        }
                  }
            }

            return true;
   }}
```

Amb un component *spinner* mostrarem la llista de moments, i iniciarem la pantalla per veure l'escena amb AR. A continuació (Figura 21) és mostra el resultat.

Figura 21 (Pantalla de selecció d'escena)

```
public class Visor4dSpinner extends Activity {

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		setContentView(R.layout.spinner);

		Log.e("MyActivity", "Spinner - inici 2");
		Spinner cmbOpciones =
			(Spinner)findViewById(R.id.cmbOpciones);

		DataBaseHelper dades = new DataBaseHelper(this);

		dades.openDataBase();

		List<String> mnuOptions;

		mnuOptions = dades.ListHistoricTime(4);

		ArrayAdapter<String> adaptador =
			new ArrayAdapter<String>(this,
				android.R.layout.simple_spinner_item,
				mnuOptions);

		adaptador.setDropDownViewResource(

		android.R.layout.simple_spinner_dropdown_item);

		cmbOpciones.setAdapter(adaptador);


		cmbOpciones.setOnItemSelectedListener(
				new AdapterView.OnItemSelectedListener() {
				public void onItemSelected(AdapterView<?>
					parent, android.view.View v, int
				position, long id) {
					}

				public void onNothingSelected(AdapterView<?>
					parent) {
					}
				});
	}
}
```

5.3.5. Aplicació framework de realitat augmentada

El framework de AR amb el què treballarem és el droidAR, com ja vàrem veure a 3.3 Framework de realitat augmentada. En aquest apartat comentarem les opcions i ens ofereix varies possibilitats de fer-ho.

Una de les condicions que demanàvem al framework d'AR era que, per resoldre la posició dels objectes virtuals adoptés una estratègia basada en la posició geogràfica. El droidAR compleix perfectament amb aquesta capacitat, però ens ofereix varies possibilitats de fer-ho.

- Coordenades geogràfiques: on el framework necessita del nostre geoposicionament, i de les coordenades en el mon real dels objectes a representar: les imatges apareixeran a les coordenades X i Y.
- Coordenades relatives: el framework marca uns eixos X,Y i Z (on normalment el punt (0,0,0) correspon a la posició de la càmera) i es representen les imatges en les coordenades d'aquest sistema de referencia.

En el nostre cas, hem de seleccionar la primera opció, ja que la segona no depèn de la nostra posició ni de l'orientació, mentre que la primera opció és ideal per posicionar les imatges sobre POI's dels quals coneixem la seva posició real. Ens ambdós casos, sempre és possible col·locar la càmera en diferents posicions per veure l'escena, simplement es tracta de posicionar-la en diferents coordenades dintre del sistema. Abans s'ha comentat que normalment la càmera és al punt (0,0,0) però podria ser tranquil·lament un (0,0,20) per una vista aèria, o un (0,20,0) per veure l'escena de lluny.

Tot seguit mostrarem les etapes per desenvolupar amb droidAR, començant per una aplicació senzilla i augmentant la seva complexitat cada cop.

5.3.5.1. Aplicació bàsica amb droidAR

La primera acció a portar a terme, és una petita configuració del nostre projecte per tal d'afegir la llibreria de DroidAR (Figura 22).

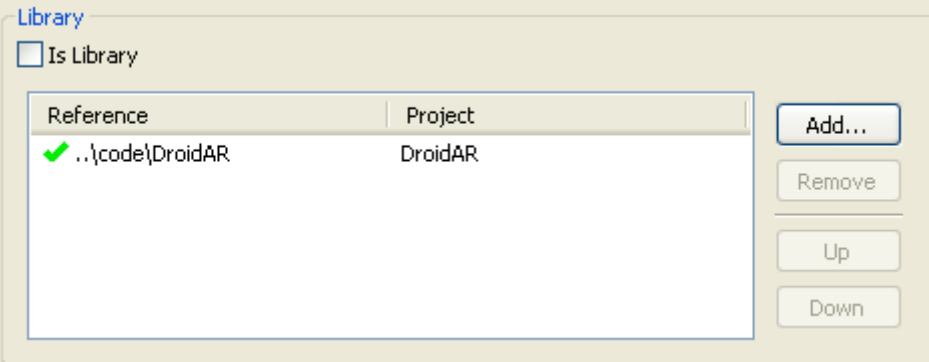

Figura 22 (Configuració del projecte per treballar amb DroidAR)

Cal modificar el fitxer **AndroidManifest** afegint noves activitats.

```
<activity
      android:name="system.ArActivity"
      android:configChanges="keyboardHidden|orientation"
      android:label="@string/app_name"
      android:screenOrientation="landscape" >
</activity>

<!-- Custom List is used to easily display lists of any type of
object -->
<activity android:name="gui.CustomListActivity" ></activity>

<!-- the error handler will be automatically displayed when an error
appears -->
<activity
      android:name="system.ErrorHandler"
      android:process=":myexeptionprocess"
      android:taskAffinity="system.ErrorHandler" >
      <intent-filter>
         <category android:name="android.intent.category.DEFAULT" />

         <action android:name="android.intent.action.VIEW" />

         <data android:mimeType="errors/myUnhandleCatcher" />
      </intent-filter>
</activity>
```

```
<!-- is needed to display information on application startup -->
<activity
      android:name="gui.InfoScreen"
      android:theme="@android:style/Theme.Dialog" >
</activity>

<!-- an activity to easily display any kind of ui -->
<activity
      android:name="gui.simpleUI.SimpleUI"
      android:theme="@android:style/Theme.Translucent" >
</activity>
```

També cal dotar de més permisos per als recursos del sistema, com són la càmera, la unitat de disc SD externa, la vibració. (Tots ells són requeriments de droidAR).

```
<uses-permission
      android:name="android.permission.CAMERA" />
<uses-permission
      android:name="android.permission.ACCESS_SURFACE_FLINGER" />
<uses-permission
      android:name="android.permission.READ_FRAME_BUFFER" />
<uses-permission
      android:name="android.permission.VIBRATE" />
<uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Finalment, cal indicar quins recursos de maquinari i programari són necessaris per el funcionament del framework.

```
<uses-feature
      android:name="android.hardware.camera" />
<uses-feature
      android:name="android.hardware.camera.autofocus" />
<uses-feature
      android:glEsVersion="0x00020000" />
```

Hem de saber que droidAR no treballa amb *layouts* clàssics d'Android, sinó que proveeix de la classe *Setup* (i les seves herències *defaultARSetup*) on s'executa tot el codi i es mostra la realitat virtual. També ens ofereix la possibilitat d'afegir controls per interaccionar amb els nostres objectes. En el nostre cas en particular (figura 23), ens interessa afegir controls per activar el GPS, un indicador de precisió de la posició (fins que no està al 100 % no comença l'activitat), i controls per jugar amb l'eix Z (alçada). Aquest últim és necessari degut a que droidAR documenta la dificultat que té el GPS d'Android per gestionar correctament les alçades, i per tant recomana afegir controls per manipular-ho manualment.

Figura 23 (Pantalla de visualització de AR)

El codi per configurar el S*etup* és molt senzill.

```
public class PositionTestsSetup2 extends Setup {

	@Override
	public void _e2_addElementsToGuiSetup(GuiSetup guiSetup,
		Activity activity) {

		guiSetup.setRightViewAllignBottom();

		guiSetup.addViewToTop(minAccuracyAction.getView());


		guiSetup.addImangeButtonToRightView(
			R.drawable.arrow_up_float, new Command() {

			@Override
			public boolean execute() {
				camera.changeZPositionBuffered(+ZDELTA);
				return false;
			}
			});
			guiSetup.addImangeButtonToRightView(
				R.drawable.arrow_down_float, new Command() {
			@Override
			public boolean execute() {
				camera.changeZPositionBuffered(-ZDELTA);
				return false;
			}
			});
	}
}
```

Dins de la mateixa classe *Setup* és on hem de programar com serà la nostra escena. Al mètode *addObjectsTo* crearem l'objecte (en aquest cas una fletxa) amb les coordenades on volem que es situï (molt a prop de la nostre habitatge habitual, coincidint amb un POI fàcil de reconèixer).

```
public void addObjectsTo(GL1Renderer renderer, World world,
      GLFactory objectFactory) {

      double Longitud = 42.342294;
      double Latitud = 3.064597;

      posA = new GeoObj(Longitud, Latitud);

      spawnObj(posA, GLFactory.getInstance().newArrow());
}

private void spawnObj(final GeoObj pos, MeshComponent mesh) {
      mesh.setPosition(new Vec(0,0,0));
      pos.setComp(mesh);
      world.add(pos);
}
```

Comprovem que el seu funcionament és correcte (figura 24), i que la fletxa apareix davant del nostre POI. Podem observar que els moviments de la càmera no alteren l'escena i, que si deixem d'enfocar el nostre POI, la fletxa desapareix fins que el tornem a enfocar. En resum el seu funcionament és correcte.

Figura 24 (Posicionar un objecte en un POI)

5.3.5.2. Presentació d'imatges i dades històriques

La nostra intenció és poder mostrar imatges, per tant hem d'investigar la manera de generar un objecte amb les imatges que tenim emmagatzemades a la nostra BD. Necessitem generar un objecte *GeoObj* al qual li puguem aplicar la capa amb les nostres fotografies, i ho farem mitjançant la classe *MeshComponent*. Qualsevol objecte de la classe *RenderableEntity* pot ser representat en el món virtual. I la classe *MeshComponent* hereta d'aquesta. A més de tenir la capacitat de ser representada, ens permet la possibilitat de crear una nova "textura" que serà l'objecte a mostrar. Veurem el resultat a la figura 25.

```
public void addObjectsTo(GL1Renderer renderer, World world,
      GLFactory objectFactory) {

      double Longitud = 42.342294;
      double Latitud = 3.064597;

      posA = new GeoObj(Longitud, Latitud);

      bm = BitmapFactory.decodeFile(DB_PATH  + DB_NAME);

      MeshComponent treangleMesh;

      treangleMesh=
            GLFactory.getInstance().
                  newTexturedSquare("nova_textura",bm);

      treangleMesh.setPosition(new Vec(1, 3, 1));
      treangleMesh.setRotation(new Vec(0, 0, 0));
      treangleMesh.setScale(new Vec(100, 100, 100));
      treangleMesh.addChild(new AnimationFaceToCamera
                                          (camera, 0.5f));


      spawnObj(posA, treangleMesh);
}
```

Figura 25 (Posicionar un objecte amb textura davant un POI)

La primera versió de la demo en la que vam treballar, no guardava les coordenades per cada imatge i per tant era impossible obtenir una representació d'una zona amb més d'una imatge. La versió definitiva (per aquest PFC, no per una versió definitiva per distribuir ja que hi ha moltes coses a millor com veurem a l'apartat 6. Conclusions) emmagatzema per cada imatge la seva localització i d'aquesta forma podem representar escenes senceres. El codi emprat és molt semblant al que s'ha presentat anteriorment per una sola imatge, simplement varia que fem un recorregut per totes les imatges donat un POI i un període de temps, i creem el objectes necessaris per la seva representació. Veiem el resultat a la figura 26.

Figura 26 (Exemple d'escena amb més d'una imatge)

```
public void addObjectsTo(GL1Renderer renderer, World world,
                  GLFactory objectFactory) {


      Visor4dDataBase dades = new Visor4dDataBase(getActivity().
                                    getApplicationContext());
                                    dades.openDataBase();

      myImages = dades.ListImage(POI_id, HistoricTime_id);

      double longitude;
      double latitude;

      MeshComponent treangleMesh;
      ByteArrayInputStream imageStream;
      Bitmap bm;

      for(int i=0; i <= (myImages.size()-1); i++) {

            longitude = myImages.get(i).longitude;
            latitude = myImages.get(i).latitude;


            imageStream = new ByteArrayInputStream(
                              myImages.get(i).blob);

            bm = BitmapFactory.decodeStream(imageStream);

            treangleMesh = GLFactory.getInstance().
                        newTexturedSquare("texture" + i,bm);

            treangleMesh.setPosition(new Vec(0, 0, 0));
            treangleMesh.setRotation(new Vec(0, 0, 0));
            treangleMesh.setScale(new Vec(10,10,10));
            treangleMesh.addChild(new AnimationFaceToCamera(
                                    camera, 0.5f));


                  posA = new GeoObj(latitude, longitude);

                  spawnObj(posA, treangleMesh);
      }

      dades.close();
}
```

El fet de veure la representació de les nostra imatge en pantalla no fa res més que generar encara més dubtes perquè l'experiència d'usuari podria ser molt millor. De fet hi ha tres factors que creiem importants.

1. Precisió: les dades dels POI les hem recopilats de vàries fonts d'informació: *Google maps* i *Google Earth* principalment. Si observem la posició al *Google maps* de l'església de Garriguella obtenim una informació, que si fem servir en el nostre projecte durà a terme imprecisió al mostrar la capa d'AR, ja que la informació sembla esbiaixada. A la següent figura (27) es veu la diferencia de la informació extreta de *Google maps* des d'un PC i la informació que mostrem nosaltres sobre el *MapView* del nostre dispositiu.

Figura 27 (Exemple de problema amb la precisió del GPS)

Aquest és un problema molt greu, ja que la nostra aplicació es basa en dades geogràfiques, si aquestes són errònies, o bé ho és la seva representació farà que la col·locació de la nostra capa d'AR no sigui tant precisa com ens agradaria.

2. Distància: la imatge que es mostra està pressa a una distancia concreta, si la representació sobre el món real no està a escala, la imatge no queda quadrada amb la realitat, obtenint una realitat augmentada que no aporta cap valor. Ho arranjàrem de forma manual, afegint controls per poder fer zoom's de les imatges presentades.

3. Perspectiva: no és el mateix veure una imatge des del punt A quan aquesta ha estat presa des del punt B. Veiem un exemple d'aquest problema a la figura 28.

Figura 28 (Diferents perspectives de la catedral de Girona)

El primer problema sembla de difícil solució, no per aquesta demo on podem obtenir les coordenades adequades de forma manual (presentant-nos al lloc d'interès) sinó per la confiança que transmet la informació de la que disposem. Hom confiava que aquestes tecnologies tindrien una fiabilitat molt més alta.

La solució als 2 darrers casos passa per conèixer la posició exacte des d'on es va prendre la imatge. El problema de la distancia quedaria arreglat, però el cas de la perspectiva encara no, ja que caldria tenir moltes més imatges, una per cada angle de visió possible que pugui tenir l'usuari. Actualment hi ha eines que permeten documentar la posició de cada imatge, però nosaltres estem tractant amb imatges de temps passats, on la dificultat de geoposicionar la informació és molt més gran. Per tant veiem de difícil solució aquest últim problema.

5.4. Proves realitzades

Anteriorment, a l'apartat 5.1 Disseny de l'aplicació, es va fer definir el joc de proves per avaluar diferents resultats. Aquests eren:

- Geoposicionament amb GPS.
- Geoposicionament amb 3G.
- Visita a un POI amb un únic moment històric.
- Visita a un POI amb més d'un moment històric.

Es pot dir que es van pensar fent dos grups de proves: el primer per comprovar amb quina tecnologia podem emprar per localitzar-nos, i el segon grup en el volum d'imatges. N'afegirem un parell més:

- Visita a un POI amb només 1 imatge.
- Visita a un POI amb més d'una imatge.

Veurem els resultats obtinguts en cadascun d'ells:

- Geoposicionament amb GPS:

  L'execució de la demo amb el GPS és la solució idònia, no per la part on es fa servir el component *MapView*, sinó per l'obligatorietat del framework d'AR de tenir aquest servei actiu. Tot i així, encara hi ha certes desviacions per la imprecisió en el moment de la localització.

  Les proves d'AR han estat un veritable problema, ja que és un framework que treballa amb el GPS i sovint aconseguir una bona senyal a l'interior d'un edifici és un problema. Les proves s'havien de fer in situ i transportar les eines de programació per desenvolupar ràpidament.

- Geoposicionament amb 3G:

  No és una opció vàlida per el nostre objectiu. Ja hem comentat anteriorment la necessitat del droidAR del GPS, per tant l'hem de descartar. Hem de tenir en

compte que la precisió de la localització mitjançant 3G no és tant acurada com la del GPS, per tant l'hauríem de descartar igualment.

Respecte als dos casos anteriors, comentem que necessitem de tenir connectivitat 3G i Wifi per el que són dades, es a dir, descartem el 3G com a eina de localització però igualment és necessari en el nostre projecte. Pensem que la majoria de POI es troben a l'aire lliure on és difícil de disposar d'una xarxa Wifi.

- Visita a un POI amb un únic moment històric.

L'objectiu d'aquesta prova era testejar l'elecció de moments històrics en el cas més desfavorable, amb només 1. La prova ha estat un èxit.

- Visita a un POI amb més d'un moment històric.

En aquest cas hem testejat l'elecció de moments històrics en un cas on n'havia fins a 7 de diferents. La prova ha resultat exitosa.

- Visita a un POI amb només 1 imatge.

Al tractar-se de representar només 1 imatge no teníem problemes de solapaments entre imatges. Ens hem trobat amb el problema de la mida de les imatges, ja que eren arxius pesants i ens provocaven un error de memòria, per tant hem hagut de treballar amb imatges de menor mida i en format més comprimit (*.png*) .

- Visita a un POI amb més d'una imatge.

Hem aplicat les millores en la mida de les imatges de l'apartat anterior. En aquest cas és quan es gaudeix del framework d'AR. Ja que a una certa distància les imatges queden solapades entre elles, a causa de la seva proximitat, és a mesura que ens desplacem cap a l'objectiu quan aquestes es van separant, col·locant-se sobre les seves coordenades.

6. Conclusions

Els SIG fa molt de temps que estan entre nosaltres, però fa relativament poc que s'ha popularitzat el seu ús gràcies (sobretot) a l'evolució dels dispositius mòbils. En aquest PFC hem treballat amb telefonia mòbil amb el sistema operatiu **Android** per desenvolupar un servei basat en la localització i aplicant també una de les tècniques més revolucionaries actualment: la realitat augmentada (AR). El resultat ha estat una petita aplicació a mode de demostració què no és res més que un petit embrió del que es pot fer amb aquest conjunt de tecnologies.

La primera part del PFC ha estat d'investigació, de conèixer millor tota la tecnologia disponible i triar l'ideal per cada problema que se'ns presentava. Creiem que la part més atractiva i la novetat d'aquest treball era el framework d'AR perquè necessitàvem una eina capaç de funcionar amb geoposicionament i la sensació és l'hem encertat de ple: **droidAR** és (potser) un gran desconegut, però amb una gran capacitat. Hem invertit una gran part de la feina en llegir documentació i provant els diferents frameworks disponibles, i no podem estar més contents de l'elecció feta.

La segona fase del projecte era el disseny i desenvolupament. El disseny ha estat bastant senzill, és una aplicació molt limitada, però tot i això el desenvolupament ha estat tediós. La poca pràctica amb el llenguatge de programació Java i les eines d'emulació d'**Android**, han fet que invertís un temps important en una aplicació sense un gran nombre de línies de codi i sense gaire complexitat. Cal comentar que, un altre cop el **droidAR** és el culpable de la majoria del temps de desenvolupament, la llibreria costa d'entendre i, fins que hem estat capaços d'entendre la filosofia, el ritme va ser molt lent.

Les proves han estat un gran problema. La incapacitat de droidAR de treballar sense GPS i de no funcionar en interiors, van ser causa d'una gran pèrdua de temps. De la

mateixa manera els dispositius mòbils requereixen d'un temps inicial per poder localitzar-nos i és un temps que s'aplica a moltes proves. Aquesta és una part a millorar de cares a futurs desenvolupaments. La documentació de les mateixes és un problema, ja que no podem gravar el resultat amb el nostre propi dispositiu, sinó que cal disposar d'un dispositiu extern, d'aquí la mala qualitat de les imatges de les proves realitzades.

El resultat final es una demostració molt senzilla que compleix amb l'objectiu del projecte i que posa les bases per a una gran millora. A la memòria s'han exposat els punts a millorar i que seran tractats més a fons a l'apartat següent 7. Visió de futur, ara només volíem comentar que a l'inici no érem capaços de preveure les possibilitats i les dificultats per orientar, posicionar, tenir en compte perspectives, etc.

Conclourem dient que el conjunt de tecnologies emprades és molt potent per desenvolupar aplicacions d'aquest tipus i que la capacitat de millora és molta. Simplement cal entendre bé els conceptes i com els apliquen les diferents eines que tenim a la nostra disposició.

7. Visió de futur

Durant el transcurs del desenvolupament d'aquest PFC, hem detectat millores a fer el futurs desenvolupaments. A continuació presentarem el que creiem han de ser les línies mestres a seguir:

- *Refactoring* del codi font: el codi escrit és millorable. S'han fet accessos redundants a la base de dades i la gestió de memòria és clarament superable.
- Usabilitat: s'ha de millorar l'experiència d'usuari, amb més missatges emergents, i representant més informació sobre el mapa.
- Orientació de la visita: la visita s'inicia en estar a menys de 50 metres d'un POI, però no es té en compte l'orientació d'aquesta. Podríem estar en el radi indicat però a la façana posterior del POI, i no és la mateixa imatge per aquesta banda que per la façana davantera, per exemple.
- Perspectiva: les imatges representades haurien de variar depenent de l'angle de visió de l'usuari respecte al POI.
- Escala de les imatges: la mida de les imatges hauria de variar, depenent dels moviments de l'usuari. Més grans a prop del POI i més petites lluny del mateix.
- Tractament d'imatges: les imatges presentades, haurien de tenir fons transparents, haurien d'estar el més "netes" possibles. Sinó la sensació és la de tenir un foto per sobre l'escena, i s'hauria de cercar una experiència millor.
- Presentació d'informació: droidAR permet posar qualsevol tipus d'informació, per exemple text i permet interactuar amb els objectes (fent

click per exemple). Són opcions que es poden aprofitar per millorar la informació.

- Gravació de visites: actualment és impossible gravar amb un mateix dispositiu la visita, hem de recórrer a dispositius externs.
- Representació 3D: droidAR, és capaç de presentar objectes 3D. Per nosaltres el gran punt a tractar.

8. Bibliografia

Bellón, S; Creixell. J; Serrano, A (2010-2011). Look!: Framework para Aplicaciones de Realidad Aumentada en Android.
http://www.lookar.net/

Botella, A.; Muñoz, A.; Olivella, R.; Olmedillas, J.; Rodríguez, J. (2009). Sistemes d'informació geográfica i geotelemàtica.

Deborah Boyer (2010). GIS and digital history.
http://www.slideshare.net/Azavea/2010-09-14gisdigitalhistory

Ed Burnette (2010). Programación Android. Madrid: Editorial Anaya.

Hoel, E; Gillgrass, C; McGrath, M (2010). History of GIS The Commercial Era: 1980 to 2010.

Janis Kampars (2009). New generation enterprise geographic information systems. Riga: Environment. Technology. Resources.

Jose Antonio Anta. Desarrollo aplicaciones móviles. Seminarios Esri España.
http://www.slideshare.net/ESRI/seminario-introduccin-a-los-sig-y-desarrollo-de-aplicaciones

Laia Descamps-Vila. Visualització 4D a través de mòbils Android 05.032-TFC-SIG Curs 2011-12. 2on Semestre

Ming-Hsiang Tsou. Integrated Mobile GIS and Wireless Internet Map Servers for Environmental Monitoring and Management

Pérez, A.; Bataller, A.; Beneito, R.; Sáenz, N; Vidal, R. (2008). Treball final de carrera.

[1]**Ronald Azuma**. A survey of augmented reality.
http://www.cs.unc.edu/~azuma/ARpresence.pdf.

Salvador Gómez Oliver (2011). Curso programación Android.

Simon Heinen. DroidAR Augmented Reality Framework
http://code.google.com/p/droidar/

Tobias Domhan (2010). Augmented Reality on Android Smartphones.

Tobias Domhan. AndAR - Android Augmented Reality.
http://code.google.com/p/andar/

Yu, M; Zhang, J; Li, Q; Huang, J. Mobile geoinformation services – Concept, Reality & Problems.

Hem consultat les següents pàgines web:

http://android.martinpearman.co.uk/b4a/osmdroid/documentation/native_android_library/org/osmdroid/views/overlay/MyLocationOverlay.html
http://androidrentable.blogspot.com.es/p/app-inventor.html
http://developer.android.com/index.html
http://dl-ssl.google.com/android/eclipse/
http://en.wikicode.org/index.php/Get_image_from_a_blob_field_android
http://es.wikipedia.org/wiki/Android
http://es.wikipedia.org/wiki/Desarrollo_de_Programas_para_Android#App_Inventor_para_Android
http://en.wikipedia.org/wiki/Distributed_GIS#Mobile_GIS
http://en.wikipedia.org/wiki/Google_Maps
http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software
http://es.wikipedia.org/wiki/Sistema_de_Informaci%C3%B3n_Geogr%C3%A1fica
http://es.wikipedia.org/wiki/Tel%C3%A9fono_inteligente
http://naturalprogression.wordpress.com/2011/07/19/samsung-galaxy-s2-adb-device-driver/#
http://programandoideas.com/ns-basic-app-studio-android-iphone/
http://searchnetworking.techtarget.com/definition/location-based-service-LBS
http://sqliteadmin.orbmu2k.de/
http://www.acolita.com/aplicaciones-sig-gis-para-smarthphone-android/
http://www.ajpdsoft.com/modules.php?name=news&file=article&sid=383
http://www.arumeinformatica.es/utiles/buscar-coordenadas-gps-en-google-maps/
http://www.eclipse.org
http://www.iqengines.com/
http://www.maclasa.com/coordenadas/index.htm
http://www.oracle.com/technetwork
http://www.reigndesign.com/blog/using-your-own-sqlite-database-in-android-applications/
http://www.sqlite.org/
http://www.tutorialforandroid.com/2009/10/how-to-insert-image-data-to-sqlite.html
http://www.webmapsolutions.com/mobile-gis-2
http://www.xatakandroid.com/programacion-android/cuatro-alternativas-a-java-para-programar-en-android