Sistema de navegació geogràfica de tipus Google Earth per a entorns dinàmics

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Sergi Villanueva Canals
i dirigit per
Lluís Ribas Xirgo

Bellaterra, 18 de Setembre de 2009

Universitat Autònoma de Barcelona

etse

Escola Tècnica Superior d'Enginyeria

El sotasignat, Lluís Ribas Xirgo

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA :

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Sergi Villanueva Canals.

I per tal que consti firma la present.

Signat: .....

Bellaterra, 18 de Setembre de 2009

Índex

Capítol 1.

Introducció

1.1. Context del problema

Avui en dia, la possibilitat de navegació visual en entorns dinàmics com ara plantes de producció està esdevenint una opció molt interessant per tal de dur a terme un monitoratge d'una situació des d'un ordinador i actuar en conseqüència sense necessitat de ser-hi present. Si considerem entorns canviants, on hi ha un alt grau de dinamisme, la supervisió i control d'aquests sistemes esdevé una necessitat. Sovint, aquesta navegació convé que sigui remota degut a la pròpia naturalesa de l'entorn. La navegació visual que es pot dur a terme utilitzant els computadors actuals pot millorar la capacitat de supervisió degut a l'àmplia informació que ofereixen i la rapidesa en la seva visualització. El problema que se'ns presenta en aquest projecte implica l'estudi de la construcció d'un sistema que permeti a l'usuari navegar en un escenari de forma virtual. Necessitem, a més, que aquesta navegació per l'escenari sigui el suficientment ràpida, precisa i còmode, és a dir, que l'usuari pugui visualitzar l'entorn amb detall i dintre d'uns marges de temps preestablerts des de que la càmera obté la imatge de l'escena fins que l'usuari la visualitza i hi navega des de l'ordinador.

Per tal de dur a terme aquest projecte, existeixen tecnologies de base que permeten la navegació en entorns reals, o virtuals, com ara eines basades en GPS. També hi ha la possibilitat de fer servir la tecnologia OpenGL, que permet desenvolupar aplicacions de gràfics en 3D i crear el nostre propi navegador. Tot i així, en aquest projecte volem crear una aplicació base sobre la que treballi el navegador geogràfic Google Earth per tal de

veure quin pot arribar a ser el rendiment d'utilitzar aplicacions que interactuin entre elles en el cas de Google Earth. Sabem que el fet d'utilitzar aplicacions que interactuin amb Google Earth afegirà una sobrecàrrega enfront de les altres alternatives, que ofereixen una solució directa. D'altra banda, la nostre solució serà més fàcil de desenvolupar i el manteniment posterior serà més senzill.

Per tant, el que pretenem en aquest projecte és estudiar la viabilitat d'utilitzar Google Earth per a la navegació per entorns dinàmics captats per una camera. Aquest navegador és capaç de traduir codi escrit en el llenguatge de marques KML que representa dades geogràfiques, cercar les imatges emmagatzemades en una base de dades geogràfica i situar-ho tot en un entorn visual on l'usuari hi pot navegar. Així, el llenguatge KML descriu punts geogràfics i els ubica sobre una quadrícula al món real. De la mateixa forma que un navegador convencional llegeix codi en llenguatge HTML i el tradueix a pàgines web, un navegador visual interpreta codi en llenguatge KML i el representa sobre el mapa del món real. Per la seva banda, una base de dades geogràfica no és més que una base de dades que permet guardar, indexar, consultar i manipular informació geogràfica. Per al nostre cas, en aquesta base de dades geogràfica es desaran imatges captades per una càmera.

El principal inconvenient que es pot trobar utilitzant aquest sistema de visualització és que ha sigut creat per tal de que funcioni en entorns estàtics. De fet, l'eina Google Earth és un navegador visual de KML, el qual llença consultes a una base de dades geogràfica que conté les imatges del món fotografiades des d'un satèl·lit. Aquestes imatges són guardades cada cert temps, és a dir, s'actualitzen, però amb una freqüència baixa ja que, òbviament, no és necessari obtenir imatges de forma constant si aquestes no sofreixen canvis de rellevància per al context en qüestió. En el nostre context això no és cert, ja que precisament l'entorn que volem visualitzar canvia constantment. A més, volem que aquests canvis puguin ser explorats des del navegador.

1.2. Objectius

En el nostre context, i seguint amb l'exemple d'una planta de producció, l'entorn a visualitzar canvia constantment. Així, el grau de dinamisme, o la freqüència d'actualització de les imatges de l'entorn pel qual l'usuari navega, ha de ser força elevat. La situació ideal es donaria amb una visualització de 25 frames per segon, que és la màxima freqüència que l'ull humà pot processar. Els requisits que es presenten davant la navegació per una planta de producció fan que l'usuari hagi de veure en el navegador visual unes imatges molt recents en el temps.

Un altre exemple que il·lustraria una situació on necessitem un seguiment en temps real seria una zona d'activitats logístiques on es gestionen fluxos de transport, tant administrativament com físicament. Aquí el nivell de dinamisme podria ser molt elevat, i es podria necessitar una exploració acurada per tal d'evitar embussos, distribuir càrregues, etcètera.. En aquest escenari sembla important arribar a tenir una visualització de 25 frames per segon. De fet, una freqüència menor, tal com 10 frames per segon, podria resultar també vàlida ja que l'usuari que està realitzant la navegació per l'escena podria actuar a temps en cas de percebre una situació que requereixi alguna acció per la seva part.

Per tant, podríem definir l'objectiu principal com l'avaluació d'un sistema que hagués de mostrar les imatges captades per una càmera que està filmant un entorn canviant sobre l'aplicació Google Earth. Es tractaria d'estudiar com es podria aprofitar aquesta aplicació i analitzar el rendiment que obtenim utilitzant-la per a navegar per un entorn dinàmic. S'hauria d'avaluar si és possible arribar a visualitzar l'entorn a una freqüència de 25 frames per segon.

Això implicaria construir una aplicació d'avaluació per obtenir el rendiment que ofereix Google Earth per al nostre projecte. Així, definim un seguit de subobjectius per a dur a terme aquesta tasca que serien els següents:

- Decidir l'arquitectura del sistema o l'aplicació base sobre la que treballarà Google Earth.
- Construir una plataforma d'avaluació d'acord amb l'arquitectura dissenyada.
- Realitzar una o vàries implementacions d'aquest sistema.
- Avaluar el rendiment de les diferents implementacions.
- Decidir la viabilitat d'aquest tipus de sistema un cop conegut el rendiment.

Per tal de complir aquest objectiu, s'utilitzarà un sistema on hi intervenen 3 elements principals:

- Una càmera que gravarà l'escenari que necessitem explorar.
- Una base de dades geogràfica que emmagatzemarà les imatges captades per la càmera.
- El navegador Google Earth per a visualitzar dades geogràfiques definides en format KML.

Una càmera estarà constantment filmant l'escenari, sent aquesta gravació emmagatzemada en forma d'imatges residents un una base de dades geogràfica.

Per últim, l'usuari podrà utilitzar Google Earth, que interactuarà amb la base de dades geogràfica que conté les imatges captades per la càmera.

Així, podem fer-nos una idea de quin serà el flux de processos que seguirà la nostre aplicació:

- Situar la càmera en el lloc adient i amb el correcte enfocament.
- Gravar la seqüència d'imatges captades per la càmera.
- Guardar les imatges en la base de dades geogràfica GIS.
- Assignar a les imatges coordenades món.

- Crear l'aplicació que muntarà codi KML amb accés a la base de dades GIS.
- Navegació per les imatges en el navegador visual de Google Earth.

Això ens duu a realitzar tot un seguit de tasques per tal de complir amb aquests subobjectius:

- Anàlisi i comprensió del problema plantejat i els fets que el motiven (15 hores)
- Llegir documentació KML (15 hores)
- Llegir documentació PostGIS (15 hores)
- Llegir documentació entorn Google Earth (10 hores)
- Instal·lació de software PostreSQL + PostGIS i Google Earth (1 hora)
- Instal·lació d'una càmera (1 hora)
- Anàlisi i disseny del sistema per traduir el vídeo de la càmera en imatges (15 hores)
- Creació del sistema per traduir el vídeo de la càmera en imatges (10 hores)
- Anàlisi i disseny de la base de dades GIS (5 hores)
- Creació de la base de dades GIS (5 hores)
- Anàlisi i disseny de la aplicació que muntarà el codi KML (40 hores)
- Creació de la aplicació que muntarà el codi KML (30 hores)
- Proves (50 hores)
- Refinament (60 hores)
- Reunions amb el director de projecte (20 hores)
- Documentació del treball realitzat (100 hores)
- Elaboració final de la memòria del projecte (20 hores)

La suma d'hores total necessàries a priori per tal de executar totes aquestes tasques estaria al voltant de les 412. Hi ha un marge de fins 38 hores fins a les 450 que seria el màxim de temps disponible per al projecte.

1.3. Planificació

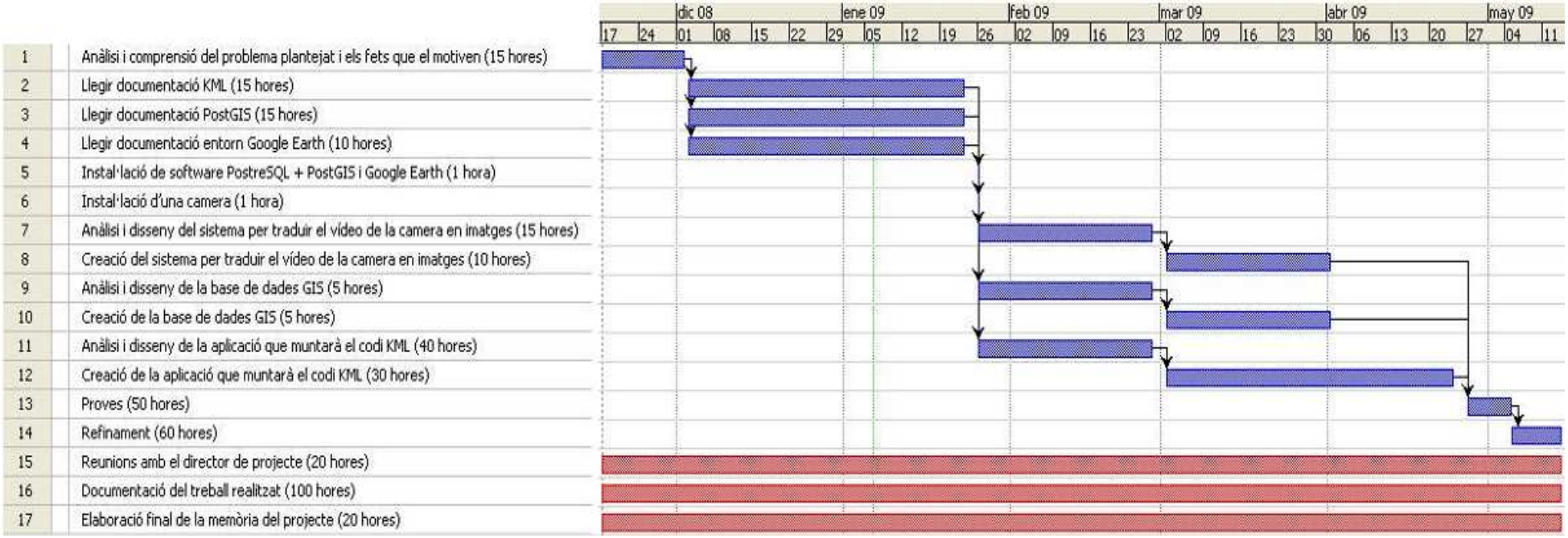

Figura 1.1

1.4. Avaluació de costos

El cost econòmic principal d'aquest projecte residiria en les màquines client i servidor que es necessiten per avaluar el projecte que volem desenvolupar. A més, necessitem disposar d'una camera per a enregistrar l'escenari que volem visualitzar. Pel que fa al software que es farà servir durant aquest projecte, es pretén que aquest estigui sempre sota la llicència de programari lliure i, per tant, no suposaria cap cost econòmic.

1.5. Organització de la memòria

Seguint a aquesta introducció, en el capítol 2 es repassarà quin és l'estat de l'art i les tecnologies disponibles relacionades amb el problema proposat, estudiant a fons la tecnologia Google Earth i les possibilitats que ofereix per al nostre projecte.

Seguidament, en el capítol 3, s'explicarà de forma detallada el desenvolupament del sistema proposat com a aplicació base sobre la qual avaluar les prestacions de Google Earth en el nostre entorn dinàmic. En el capítol 4 es mostraran els resultats obtinguts amb el sistema proposat, comentant varis casos d'estudi i avaluant-ne resultats. El capítol 5 contindrà les conclusions, amb un resum dels resultats més significatius així com línies possibles de continuació.

Per a finalitzar el document, trobarem les referències bibliogràfiques que s'han fet servir al llarg del desenvolupament d'aquest projecte i, finalment, els apèndixs.

Capítol 2.

Estat de l'art i de les tecnologies

2.1. Google Earth i Internet

En el capítol 1, hem descrit quina serà l'arquitectura del sistema que resoldrà el problema plantejat, indicant quins són els elements que hi prendran part. Ara, més concretament, podem diferenciar entre els elements que tracten i proporcionen la informació final i aquells elements que treballaran amb aquesta informació generada. Per tant, estem davant d'un escenari d'aplicació client-servidor, on el nostre servidor proporciona informació i el nostre client la interpreta i treballa amb ella. Més concretament, el client estarà format per un usuari que navegarà a través del navegador Google Earth que interpretarà la informació generada pel sistema que actuarà com a servidor.

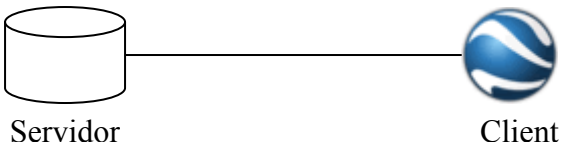

Figura 2.1

Cal remarcar que la tecnologia que utilitzem per a la navegació, Google Earth, està sent cada cop més utilitzada arreu d'Internet per a desenvolupar tot tipus d'aplicacions de sistemes cartogràfics o *SIG* (Sistemes d'Informació Geogràfica). És important que el nostre projecte estigui ben definit dins el context d'Internet i Google Earth per tal d'assegurar la compatibilitat amb altres projectes que puguin estar relacionats. Així, necessitem que els fitxers resultants del nostre projecte puguin ser sempre interpretats per un navegador geogràfic estàndard, com ara Google Earth.

De fet, l'èxit dels nous Sistemes d'Informació Geogràfica i les aplicacions que estan sorgint al voltant d'ells és degut, en gran part, a l'aparició de Google Earth. Si aconseguim que la aplicació que desenvolupem en aquest projecte sigui totalment interoperable amb aquestes aplicacions mitjançant Internet, podríem definir comunicacions entre elles que aportin noves funcionalitats i projectes més complexos.

D'aquesta forma, es pretén que el resultat del sistema que definim en aquest projecte sigui interpretable a través d'aplicacions que utilitzen Internet i Gogle Earth, amb la avantatge de que aquest sistema garanteix a l'usuari un caràcter estàndard en les dades que s'utilitzen.

2.2. El llenguatge KML

Com hem comentat anteriorment, el llenguatge en el que es basarà el nostre projecte és l'anomenat KML (*Keyhole Markup Language*). Tal i com el seu nom indica, KML és un llenguatge de marques que està basat en XML (*eXtensible Markup Language*) i que representa dades geogràfiques en tres dimensions. El llenguatge va ser creat per la empresa *Keyhole Inc*, que va ser adquirida per *Google* l'any 2004. El navegador Google Earth va ser la primera aplicació capaç d'interpretar i editar arxius KML.

Un llenguatge de marques combina dades i etiquetes que defineixen aquests dades. Tal i com hem comentat, KML està basat en XML, un llenguatge de marques que permet definir la gramàtica de llenguatges específics, és a dir, per a diferents necessitats per tal d'afavorir l'intercanvi d'informació estructurada entre diferents plataformes. KML, particularment, defineix una gramàtica i un format de fitxer XML per a la modelització i l'emmagatzematge d'elements geogràfics per tal de poder visualitzar-los en un navegador capaç d'interpretar fitxers KML, com ara Google Earth.

Un document KML serà vàlid si el seu contingut respecta les regles definides en el *Document Type Definition (DTD)* definit per al llenguatge KML. Per tant, els elements que formen el document hauran de seguir unes regles per tal de que aquest sigui vàlid i ben format. Aquests elements s'organitzen jeràrquicament, de forma que en cada document sempre hi ha un sol element arrel, que en el cas del llenguatge KML és l'element <kml>. A partir d'aquest element es defineixen els subelements lògics en forma d'arbre que construeixen el document KML que serà interpretat pel navegador, seguint sempre l'estructura definida en el *DTD*.

2.3. El navegador Google Earth i KML

2.3.1. Mostrar una imatge en el navegador

L'eina principal per a l'usuari en el nostre sistema serà el navegador Google Earth. Aquesta és la interfície per a l'usuari d'un sistema que realitza una tasca semblant a la que estem tractant: representar imatges del món captades per satèl·lit sobre la superfície de la Terra i mostrar-ho a través del seu navegador. D'aquesta forma, el producte final generat per la nostre aplicació ha de ser entès per un navegador de l'estil Google Earth, és a dir, ha de ser un document que compleixi amb l'estàndard definit en el llenguatge KML. Amb això ja hem definit globalment quin ha de ser el resultat que ofereixi la solució al nostre problema de visualització. Des del punt de vista de l'usuari, aquest monitoratge l'ha de

poder dur a terme obrint la aplicació Google Earth i fent que aquesta llegeixi el document KML que hem generat.

Per tant, una primera tecnologia a estudiar des de l'inici del desenvolupament d'aquest projecte és el llenguatge KML. Així, cal estudiar-ne la documentació que es pot trobar al web de Google Earth i veure quines possibilitats ofereix per a la resolució del nostre problema, pel que fa a la representació d'imatges proporcionades per nosaltres sobre el navegador. Una solució que se'ns ofereix és la possibilitat d'incorporar una capa amb una imatge donada pel programador sobre el món utilitzant el sistema de referència de coordenades (longitud, latitud i, opcionalment, altitud). Per tant, podem construir un document KML que carregui una imatge, la situï en una capa i la mostri sobre la superfície allà on li indiquem. Ara sabem que és possible fer que el navegador mostri unes imatges diferents de les que mostra per defecte, unes imatges que nosaltres proporcionem en el moment que vulguem.

2.3.2. Estructura client-servidor

Fins ara, hem tractat un escenari amb informació de caràcter local, és a dir, un navegador que obre un document KML que tenim en el nostre ordinador. Aquest, però, no és exactament el que ens trobem en el nostre context. Al treballar amb un sistema client-servidor, el client ha de ser només responsable de navegar sense preocupar-se de generar el codi que li permet dur a terme aquesta navegació. Per tant, cal esbrinar si la tecnologia KML proporciona algun mecanisme per tal de que l'usuari final pugui treballar amb informació generada pel servidor. Una possible solució seria que el client obrís un document KML que simplement s'encarregués de comunicar-se amb el servidor i que aquest li proporcionés el codi que ha d'interpretar Google Earth. Veurem més endavant com això és possible amb l'ajut de llenguatges d'*scripting*, com ara *PHP* o *Phyton*.

2.3.3. Dinamisme

El fet de que el navegador mostri la imatge que nosaltres proporcionem des del servidor és un requisit indispensable per a la nostre aplicació. No obstant, si no podem recarregar constantment aquestes imatges sobre el navegador d'alguna manera, de poc ens servirà emprar aquest navegador per al nostre context. Així, la pregunta que cal fer-nos és si és possible que el navegador accepti constantment documents KML en un espai curt de temps i quin és exactament aquest temps.

En la documentació que ofereix Google, veiem que existeix la possibilitat de refresc d'una imatge definit amb un nombre real, que equival als segons que transcorren entre refresc i refresc. Per tant, sembla viable obtenir un refresc constant en les imatges que el navegador mostra. Recordem que aquest fet és molt important per al nostre projecte ja que l'entorn que volem explorar des del navegador Google Earth té un alt nivell de dinamisme.

2.3.4. Comunicació bi-direccional

Per últim, ens queda un requeriment que la tecnologia KML ha de complir per tal de que es puguin satisfer les necessitats del nostre sistema. Fins ara la comunicació ha estat unidireccional, és a dir, la informació ha anat de la nostre aplicació cap al navegador de Google Earth, on ha estat interpretada. No obstant, necessitem que el navegador també enviï informació cap a la nostre aplicació: les coordenades del visor. Així, el navegador enviarà a la aplicació quines coordenades està veient l'usuari a la pantalla cada cop que vulgui actualitzar el contingut del que s'està mostrant. El sistema haurà de ser capaç de proporcionar una nova imatge tenint en compte aquestes coordenades. Concretament, s'haurà de generar un nou document KML que contingui una capa amb la nova imatge que l'usuari ha de visualitzar en el navegador, tenint en compte les coordenades del visor actuals. Podem veure un esquema d'aquesta situació a continuació, en la figura 2.2.

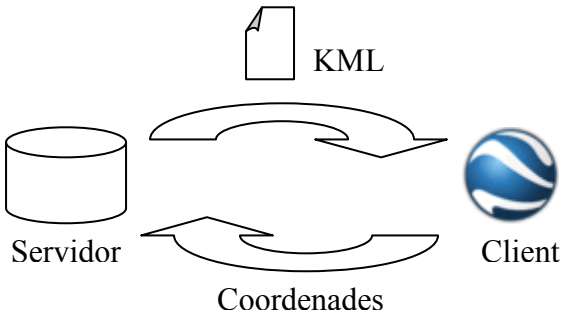

Figura 2.2

Necessitem, doncs, que la tecnologia KML, juntament amb el navegador de Google Earth, permetin aquest bi-direccionalitat de la informació. Consultant la documentació que Google ofereix sobre KML, veiem que hi ha una manera d'aconseguir-ho. Es tracta d'utilitzar el que Google anomena *View-Based Refresh Queries*, és a dir, un sistema de refresc en el que el propi navegador proporciona les coordenades de la regió que l'usuari està veient en un moment donat. Amb aquesta tecnologia podrem enviar constantment les coordenades del visor a la nostre aplicació, que les interpretarà i generarà noves imatges per a les coordenades en qüestió. D'aquesta manera, aconseguim que l'esquema de dalt sigui cíclic i *at eternum*, és a dir, el client envia constantment les coordenades al servidor, que sempre romandrà a l'espera d'aquestes per generar nou codi.

2.3.5. *Scripting* i *KML*

Fins aquí, veiem que la utilització de Google Earth sembla factible per al nostre projecte. Podem generar una comunicació bi-direccional entre un client Google-Earth i un servidor que enviarà codi cap al client. Així, al client necessitem un codi KML que es pugui obrir amb el navegador de Google Earth i que incorpori un sistema de refresc per tal

d'actualitzar-se constantment. Tal i com hem comentat anteriorment, descobrim gràcies a la documentació de KML publicada per Google, que hi ha una forma de fer que el codi del client apunti a un document situat en un altre màquina, que en el nostre cas serà el servidor. Aquest document pot ser codi KML o un script escrit en qualsevol llenguatge d'scripting. Gràcies a aquest script, el servidor serà capaç de proveir el nou codi per al client. Aquest link cap a un altre document s'anomena Network Link en KML, i té associat, entre d'altres propietats, un interval de refresc que serà el que ens permetrà executar l'script del servidor constantment per tal d'obtenir noves imatges. Podem veure un esquema global del sistema a la figura 2.3.

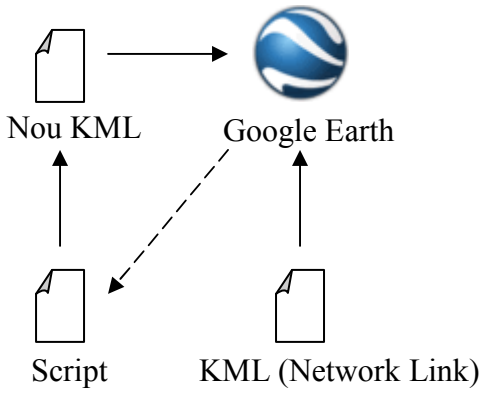

Figura 2.3

Així, si combinem el Network Link amb una tecnologia d'scripting, podem generar codi dinàmicament, que és el que ens interessa en el nostre projecte. Per tant, sabem que el client disposarà d'un Network Link que apuntarà a un script al servidor que generarà el nou codi, tal i com veiem a la figura anterior. Aquest script rebrà unes dades i generarà el codi en conseqüència. D'entrada sembla lògic pensar que l'script rebrà les coordenades del visor del client i generarà la d'alguna manera la nova imatge. De fet, la sortida de l'script hauria de ser un document KML ben format que contingui la nova imatge per a les coordenades que ha rebut.

És important que el nostre servidor proporcioni sempre documents KML estrictament ben formats. Primerament, és possible que el navegador deixi de funcionar si no sap interpretar el codi KML que està rebent i, més important encara, volem que el nostre projecte sigui totalment vàlid dins la ontologia que abans hem comentat. Si tenim l'objectiu de fer possible la interactuació amb altres projectes que treballin amb SIG's, necessitem que els documents KML que generem siguin sempre vàlids.

2.3.6. Capturar les imatges

Ens falta descriure com aconseguirem les imatges de l'entorn que volem explorar. Necessitarem un càmera que obtingui el vídeo que està enregistrant i l'emmagatzemi d'alguna manera en una base de dades. De fet, el que emmagatzemarem seran les imatges que formin aquest vídeo, ja que amb el que treballarem finalment seran imatges, que és el que pot carregar un document KML. Per tant, hi haurà un procés que s'encarregarà de convertir el vídeo que s'està enregistrant en tot moment en imatges que es guardaran en una base de dades. Llavors, l'script que generi el nou codi haurà de comunicar-se amb aquesta base de dades i recuperar les imatges que siguin necessàries.

2.3.7. Esquema analític

Per concloure aquest capítol, podem veure en la figura 2.4 quines seran les diferents parts que intervenen en el nostre projecte i quina és la informació que es transferirà entre elles per tal d'obtenir un monitoratge constant d'un entorn dinàmic enregistrat per una càmera utilitzant Google Earth.

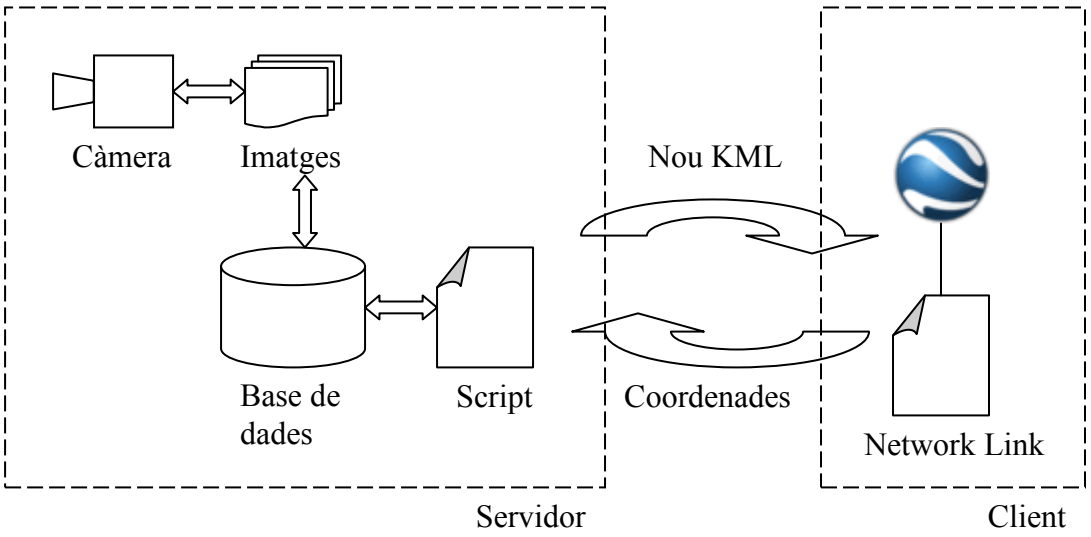

Figura 2.4

Capítol 3.

Plataforma de navegació visual

En el capítol anterior s'han descrit les tecnologies que farem servir per a desenvolupar el nostre projecte i hem descrit les diferents parts que hi participaran. Hem definit el flux d'informació entre cada element així com la funcionalitat d'aquests. En aquest capítol aprofundirem en cada un d'aquests elements, descrivint com els hem desenvolupat, i entrant en els detalls d'implementació, amb les possibles modificacions de la idea inicial i els problemes que ens hem trobat durant el camí. Així, l'objectiu del capítol és detallar com es construeix la plataforma sobre la que farem les proves d'avaluació per veure fins a quin punt és viable el nostre projecte.

3.1. El Client

Ja hem comentat anteriorment que el client necessitarà obrir un link cap al servidor per tal de comunicar-s'hi. Es tracta d'un codi KML que contindrà un element Network Link que farà referència a un script situat al servidor. Per tant, el client només necessitarà dues coses per tal de fer anar l'aplicació final del nostre projecte. Per una part, necessitarà tenir instal·lat l'aplicació Google Earth. Per a la realització d'aquest projecte, hem fet servir la

versió 4.2, però no hi hauria d'haver cap problema en utilitzar la versió més recent de Google Earth ja que el codi KML amb el que treballem seria sempre vàlid. D'altra banda, el client també necessitarà un fitxer que contingui l'enllaç cap al servidor. Aquest enllaç el detallem en el següent punt.

3.1.1. Network Link

Un element essencial en el client, tal i com hem comentat, seria el fitxer KML que conté el vincle cap al servidor. En aquest codi KML trobarem, un element Network Link que apunta al servidor, tal i com veiem a continuació:

```
<NetworkLink>
  <name>Monitoratge d'entorn dinàmic</name>
  <visibility>0</visibility>
  <open>0</open>
  <description> Crida al servidor cada 2 segons. </description>
  <refreshVisibility>0</refreshVisibility>
  <flyToView>1</flyToView>
  <link>
    <href>http://localhost/test.php</href>
    <refreshMode>onInterval</refreshMode>
    <refreshInterval>2</refreshInterval>
    <viewRefreshMode>onStop</viewRefreshMode>
    <viewRefreshTime>2</viewRefreshTime>
    <viewBoundScale>1.25</viewBoundScale>
    <viewFormat>


      <![CDATA[BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]&
      CAMERA=[lookatLon],[lookatLat],[lookatRange],[lookatTilt],[l
      ookatHeading]]>>

    </viewFormat>
  </link>
</NetworkLink>
```

Veiem com l'element <NetworkLink> té varis subelements que el defineixen, sent el més interessant l'element <link>. A continuació descrivim tots aquests elements, alguns d'ells específics de Network Link:

- name: nom que identifica l'element per a l'usuari i que apareix com una etiqueta en el navegador Google Earth.
- visibility: valor lògic que indica si l'element serà visible o no en el navegador.
- open: valor lògic que indica si el document apareixerà tancat o obert en el panell de control del navegador quan s carrega per primer cop.
- description: text que apareixerà dins un globus d'informació al navegador.
- refreshVisibility: element específic de Network Link; valor lògic que indica si cal refrescar la visibilitat d'un ítem cada cop que el Network Link es torna a carregar. Així, si deixem aquesta propietat a 1, en el cas de que l'usuari desactivi la visibilitat de la imatge que estem recarregant, quan aquesta es torni a carregar, la visibilitat tornarà a activar-se.
- flyToView: element específic de Network Link; valor lògic que indica si, a l'obrir el fitxer a on apuntem, el navegador hi navegarà cap a les coordenades on estigui establert, en cas de tenir-ne de definides.
- link: en el nostre aquest element especifica la localització d'un arxiu extern, que bé pot ser directament un arxiu KML o un script que com a resultat retorni codi KML.

Potser l'element més interessant és <link>, que té els següents subelements:

- href: indica la URL del fitxer a carregar, que bé pot ser una direcció HTTP o un fitxer local. En el nostre cas, com que el pare de l'element <link> és un element <NetworkLink>, href fa referència a un fitxer KML (o un script que retorni codi KML).

- refreshMode: aquesta propietat especifica un mode de refresc per al link que carreguem. Hi ha 3 tipus de refresc:
    - onChange: refrescar quan el fitxer és carregat i quan els paràmetres del link canviïn.
    - onExpire: refrescar només quan ha passat el temps d'expiració.
    - onInterval: refrescar cada n segons, sent n especificat en l'element <refreshInterval>. Aquest és el mode de refresc que en interessa en el nostre projecte, ja que volem que el contingut de link es recarregui constantment.
- refreshInterval: serà la quantitat de segons que esperem per a refrescar el Network Link.. Segons la documentació proporcionada per Google, aquest valor és un real. Sembla, doncs, factible que puguem ajustar la freqüència de refresc al valor que vulguem.
- viewRefreshMode: especifica com es refresca el link quan la càmera canvia, és a dir, quan l'usuari es mou per l'escenari a través del navegador. Hi ha 4 opcions diferents per a aquest mode de refresc:
    - never: ignorar canvis en la càmera. D'aquesta manera, el refresc només es donaria a terme segons l'especificat en l'element <refreshMode>.
    - onStop: refrescar el fitxer n segons després de que el moviment finalitzi, on n està especificat en l'element <viewRefreshTime>.
    - onRequest: refrescar el fitxer només quan l'usuari així ho desitgi, fent servir el navegador per seleccionar la opció de refresc immediat.
    - onRegion: refrescar el fitxer quan la regió esdevé activa, és a dir, quan la càmera està enfocant les coordenades de la regió sobre la que treballa el link. En el nostre cas sembla que aquesta és la opció adient, ja que només volem actualitzar el link, que resultarà en una imatge, quan aquesta és visible per al client des del navegador.

- viewRefreshTime: si a l'element <viewRefreshMode> hem indicat la opció onStop, aquí indiquem els segons que transcorren des de que la càmera s'atura fins al refresc de la vista.

- viewBoundScale: indica si cal fer un escalat de la imatge que mostrem per pantalla. El valor que indiquem és el factor d'escalat de la imatge; així, un valor inferior a 1 fa la imatge més petita mentre que un valor més gran que 1 faria la imatge més gran. En el nostre cas, un factor 1 fa que la imatge ocupi tota la pantalla. Es decideix aplicar un escalat de 1.25 per tal de que la imatge vagi més enllà dels límits de la pantalla degut a un petit problema experimentat al moure la càmera. Si no fem la imatge una mica més gran que la pantalla, al moure la vista d'usuari es pot arribar a veure, durant un instant, el que hi ha més enllà de la imatge, és a dir, els mapes que proporciona Google Earth. Aplicar un escalat de 1.25 és suficient per evitar aquest efecte negatiu.

- viewFormat: especifica el format de la cadena alfanumèrica que contindrà la consulta que s'afegirà al link indicat en l'element <href>. D'aquesta manera, aconseguim d'una forma ben senzilla que l'script rebi les coordenades del visor del navegador. Enviem la variable composta BBOX que conté les coordenades nord, sud, est i oest de la regió que està veient la càmera del navegador fent ús del protocol HTTP. També enviem la variable CÀMERA amb el valor lookatRange que indica la alçada en metres a la que es troba la vista actual. Veurem més endavant com l'script necessita aquesta variable per a escollir la nova imatge que mostrarem. Així, la URL final que carrega el Network Link serà:

http://SERVIDOR/NOM_SCRIPT?BBOX=[bboxWest,bboxSouth,bboxEast,bboxNorth]
&CAMERA=[lookatRange]

on w,s,e,n són les coordenades oest, sud, est i nord respectivament del visor del navegador en un moment donat.

3.2. El Servidor

Ja hem comentat que, pel que fa al client, l'únic fitxer que necessàriament disposarà per al funcionament del nostre projecte serà el fitxer KML amb el Network Link que apunta al nostre servidor. El fitxer que conté l'script que generarà codi KML que serà retornat al client, residirà en el servidor. Aquest script serà finalment codificat en el llenguatge d'scripting PHP. Per tal de que el client pugui comunicar-s'hi, també serà necessari instal·lar un servidor web que rebi peticions HTTP i enviï el codi KML. A més, en el servidor trobarem també la càmera que enregistra l'escenari així com la base de dades on hi guarda la informació que genera.

3.2.1. El servidor web

Per tal de que el servidor pugui rebre peticions HTTP procedents del client, a través de la xarxa, necessitem configurar un servidor web instal·lat en la màquina servidor. Volem un servidor web fiable i senzill ja que només es rebrà un tipus de petició i, segons els requeriments d'aquest projecte, no haurem de manegar moltes peticions al mateix temps, tot i que contínuament estarem treballant amb el client degut a la naturalesa de la aplicació.

Així, instal·lem en el servidor un paquet de programari lliure anomenat XAMPP, que conté un servidor web Apache força simple, així com les eines necessàries per a utilitzar el llenguatge d'scripting PHP, necessari també per al correcte funcionament del servidor del nostre projecte. La seva simplicitat, així com el fet que es distribueixi sota llicència pública general, són les raons principals per a la seva elecció.

3.2.2. La càmera

Per adquirir les imatges de l'entorn a explorar utilitzem una webcam que enregistrarà una escena de forma ininterrompuda. La webcam utilitzada durant el desenvolupament d'aquests projecte és una Philips SPC220NC com la que veiem en la figura 3.1 que enregistra vídeo amb resolucions que van des de 160x120 píxels fins a 1280x960 píxels amb espai de colors RGB utilitzant 24 bits, el que significa que s'utilitzen 256 bits per a definir cada component de l'espai RGB, és a dir, disposem de 256*256*256 = 16,7 milions de colors diferents.

Pel que fa a la resolució, s'haurà d'escollir quina és la més adient per al nostre projecte. Cal arribar, doncs, a un compromís adient entre la quantitat d'informació que transferim i la qualitat de la imatge final. Òbviament, com més resolució tinguin les imatges del vídeo, més informació caldrà transferir entre servidor i client i, per tant, la velocitat de l'aplicació es veurà reduïda.

Figura 3.1

Per tant, cal decidir si estem disposats a perdre velocitat per tal de guanyar qualitat en les imatges. La nostre aplicació requereix, sobretot, rapidesa en l'execució, ja que volem explorar un escenari dinàmic i volem fer-ho el més ràpid i fluid possible; per tant, sembla lògic pensar que el més adient és perdre qualitat en la imatge per tal d'accelerar la transferència d'informació i, per tant, la velocitat d'execució de la aplicació final. Per tant, escollirem inicialment una resolució baixa: 320x240 píxels.

La càmera pot capturar vídeo a una velocitat de fins a 30 fotogrames per segon. Es decideix, però, que amb 25 fotogrames per segon és suficient per a navegar per un entorn dinàmic com el que suposem en aquest projecte. De fet, si aconseguim que el client visualitzi l'entorn a una freqüència de 24 frames per segon, ja haurem aconseguit el màxim que l'ull humà pot percebre i, per tant, el màxim de fluïdesa en la visualització que es pot obtenir.

D'altra banda, cal remarcar que el nombre de frames que obtenim per segon es traduirà, a fi de comptes, amb el nombre d'imatges que guardem a la base de dades cada segon. Ens hauríem de preguntar quines són les necessitats que requereix l'entorn que estem explorant pel que fa a la freqüència de mostreig del vídeo. En el cas que hem explicat en el primer capítol, l'escenari d'una planta de producció, no sembla excessivament rellevant el nombre d'imatges per segon que enregistrem i podem concloure que amb 25 frames per segon hi ha més que suficient. Però en altres escenaris, on hi hagi un nivell de dinamisme molt alt, podria ser interessant augmentar aquesta freqüència de mostreig independentment de si el client podrà percebre o no els canvis mentre explora l'escenari amb el navegador. D'aquesta manera, tindríem enregistrat tot el que succeeix en l'escenari a la base de dades per futures consultes externes a la nostre aplicació.

3.2.3. Del vídeo a les imatges

Ja hem comentat en l'anterior capítol que el vídeo enregistrat per la webcam havia de ser traduït a imatges per tal de poder mostrar l'escena en el navegador Google Earth. Aquestes imatges hauran de residir en una base de dades per tal de ser emmagatzemades. Per tant, necessitem un procés intermedi entre la webcam i la base de dades. Aquest procés serà l'encarregat de passar el vídeo inicial a imatges. Podem veure un esquema de la situació en la figura 3.2.

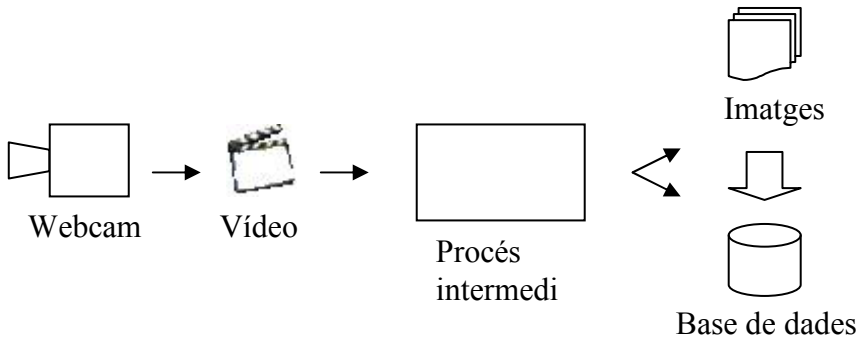

Figura 3.2

Anem a descriure quina serà exactament la tasca que durà a terme el procés intermedi entre la webcam i la base de dades. En un primer moment es decideix que la única tasca d'aquest procés sigui la de convertir el vídeo en imatges i emmagatzemar-les en la base de dades sense modificar de cap forma la imatge, tal i com hem comentat fins ara. No obstant, això comportaria una limitació per al client. El navegador de Google Earth permet fer zoom mentre el client explora les imatges, donant la sensació d'alçada, és a dir, les imatges estan captades a diferents resolucions segons l'alçada a la que es trobi el client. Aquesta alçada es pot modificar fàcilment des del navegador, amb el propi ratolí. Així, l'usuari del navegador té la sensació de poder fer zoom en les zones d'interès, per tal d'obtenir una imatge més detallada. Per aconseguir aquest efecte amb les nostres imatges, es decideix que la nostre aplicació també permeti que l'usuari pugui utilitzar el

zoom del navegador per tal d'obtenir imatges més o menys detallades. Això vol dir que tindrem imatges amb més resolució que d'altres.

Inicialment, es fixen 2 nivells de detall en les imatges, és a dir, l'usuari veurà la imatge completa en el navegador i, quan decideixi obtenir més detall en la imatge, és a dir, disminueixi l'alçada des del navegador, la imatge que es mostrarà tindrà més resolució en la zona ampliada. Per tant, caldrà definir una alçada llindar que separi els 2 nivells de resolució.

Per a donar la sensació de diferent alçada en les imatges, caldrà les imatges de menys alçada tinguin més nivell de detall que les imatges superiors. Per a les imatges de més alçada utilitzarem les imatges resultants directament del vídeo; aquestes seran dividides en quatre quadrants de mateixa mida cada un per tal d'obtenir les imatges de segon nivell d'alçada. D'aquesta manera, quan l'usuari baixi per sota l'alçada llindar, la imatge que veurà en el navegador no serà la imatge completa si no un dels 4 quadrants de la imatge, depenent de les coordenades on estigui situada la regió del visor del navegador.

Amb això aconseguim que l'usuari pugui fer zoom en una de les regions de la imatge. Tanmateix, d'aquesta manera no aconseguim que l'usuari experimenti una millora en la imatge ampliada, és a dir, en un dels quadrants de la imatge completa. Per tal d'aconseguir-ho, necessitem que la imatge completa (primer nivell) tingui menys nivell de detall que les imatges corresponents als quatre quadrants (segon nivell). Es decideix, doncs, que caldria reduir la resolució de les imatges del primer nivell per tal de que les del segon nivell tinguin més qualitat.

Per tant, l'efecte d'aconseguir imatges més detallades quan l'usuari disminueix l'alçada del navegador l'aconseguim dividint la imatge original en 4 parts i fent que aquestes parts siguin imatges amb més qualitat de detall que la imatge completa. Ja hem indicat que la resolució de les imatges del vídeo es de 320x240. Això vol dir que cada quadrant de la imatge serà de 160x120 píxels.

El que podem fer és, un cop hem retallat la imatge per obtenir els quadrants, reduir la imatge original també a 160x120 píxels. Amb això aconseguim imatges de mateixa mida però diferent qualitat, ja que a les imatges de segon nivell utilitzem els mateixos píxels per a representar una regió més petita de l'escenari. Podem veure el procés en la figura 3.3.

En un primer moment es decideix emprar l'entorn Matlab per a la creació d'aquest procés que ha de dur a terme aquestes tasques, degut a la simplicitat del llenguatge. Així, desenvolupem una petita aplicació que segueix el següent algorisme:

```
mentre (video)
     adquirir_frame;
     dividir_quadrants;
     disminuir_resolució_imatges_nivell_1;
     introduir_informació_a_BD;
fi_mentre
```

Per tant, l'aplicació desenvolupada en Matlab haurà de ser capaç d'obtenir constantment el vídeo que la webcam està capturant. Per cada fotograma que s'obtingui, s'hauran de crear 5 imatges. Una serà la imatge completa, de la qual s'haurà de reduir la resolució per tal de baixar-ne la qualitat. Les altres quatre imatges corresponen als quatre quadrants de la imatge amb la resolució original.

Podem trobar el codi complet del procés Matlab al final d'aquest document, a l'apartat dedicat als annexos.

Imatge original (320x240)

Imatge original reduida
(160x120)

Imatges dels quadrants
(160x120)

Figura 3.3

Arribats a aquest punt, cal decidir quin format tindran les 5 imatges per frame que es creen. En un primer moment, es decideix que les imatges es gravin en format *JPEG* (*Joint Photographic Experts Group*). Aquest format segueix un algorisme de compressió que fa disminuir força la grandària de la imatge i, òbviament, la qualitat de la mateixa. No obstant, aquest fet requereix que, a l'obrir la imatge, s'hagin de fer càlculs per aplicar l'algorisme de descompressió de la imatge. Així doncs, aquest format faria que la transferència de les imatges entre client i servidor fos ràpid, però requeriria un treball constant en el client per tal d'obrir-les.

Una alternativa la tenim en el format *BMP* (*BitMapP*); aquest format no comprimeix la imatge, obtenint uns fitxer força més grans que amb el format *JPEG* però amb molta més qualitat. L'avantatge, més enllà de la qualitat de la imatge, és que aquests fitxers són extremadament ràpids d'obrir i, per tant, el client no haurà de dur a terme càlculs intermedis cada cop que obri les imatges, fet molt important en el nostre projecte.

Així, la decisió d'escollir un o altre format per a les imatges amb les que treballem no és trivial. La grandària d'un fitxer amb una imatge *JPEG* pot arribat a ser fins 10 o 20 vegades més petit que la d'una imatge *BMP* sense que l'ull humà noti en excés la pèrdua de qualitat en la imatge. Tot i així, volem que el client no hagi de dur a terme càlculs abans d'obrir la imatge amb el navegador, ja que això podria disminuir molt la velocitat d'execució en el client. Així, es pren la decisió de treballar amb imatges en format *BMP*.

3.2.4. La base de dades

Ja sabem en quin format ha de guardar les imatges el procés Matlab. El següent pas que ha de realitzar és la introducció de la informació referent a cada frame en la base de dades que resideix en el servidor. Recordem que aquest procés també es troba en el propi servidor. Per tant, ens hem de connectar a una base de dades local i introduir-hi informació constantment.

Bàsicament, necessitem que la base de dades guardi, per a cada imatge, la direcció física del fitxer que conté la imatge i les coordenades nord, sud, est i oest que cobreix la imatge. La direcció física de la imatge haurà de ser accessible per a l'aplicació Apache, és a dir, el directori on estiguin les imatges haurà de ser configurat al servidor web per tal de que aquests fitxers amb imatges siguin accessibles des del client. Aquest fet és molt important ja que si hagués cap problema, el servidor web no podria oferir les imatges al client i això comportaria un error durant l'execució de Google Earth en la màquina des d'on es vol navegar per l'entorn que estem enregistrant.

Cal remarcar que, en aquest context, entenem una imatge com qualsevol fitxer resultant del procés Matlab que hem descrit anteriorment. És a dir, tant imatges completes com quadrants de cada frame. Per tant, una tercera dada que hauria de quedar reflectida en la base de dades és el nivell de detall de la imatge que, tal i com hem comentat, pot ser de primer nivell o segon nivell si es tracta d'un dels quatre quadrants en els que dividim la imatge original d'un frame donat. Crearem, doncs, dues taules en la base de dades. La primera taula, anomenada IMG_1, contindrà les dades de les imatges completes de cada frame mentre que la segona taula, que anomenem IMG_4, contindrà les dades de les imatges corresponents a cada un dels quatre quadrants de la imatge original.

Per últim, assignarem un identificador únic a cada imatge com a clau primària. D'aquesta manera, una tupla de cada una de les taules de la base de dades vindrà definida pels següents camps, tots obligatoris:

Taula IMG_1

Nom Columna	Tipus Columna	Descripció
IMG_Id (Clau primària)	Integer	Identificador únic
IMG_Path	Text	Direcció física del fitxer
IMG_North	Real Double Precision	Coordenada nord de la imatge
IMG_South	Real Double Precision	Coordenada sud de la imatge
IMG_West	Real Double Precision	Coordenada oest de la imatge
IMG_East	Real Double Precision	Coordenada est de la imatge

Taula IMG_4

Nom Columna	Tipus Columna	Descripció
IMG_Id (Clau primària)	Integer	Identificador únic
IMG_Path	Text	Direcció física del fitxer
IMG_North	Real Double Precision	Coordenada nord de la imatge
IMG_South	Real Double Precision	Coordenada sud de la imatge
IMG_West	Real Double Precision	Coordenada oest de la imatge
IMG_East	Real Double Precision	Coordenada est de la imatge
IMG_Region (Clau primària)	Small Integer	Quadrant de la imatge

També cal remarcar que es crearan índexs sobre les columnes més utilitzades, que serien IMG_Path, IMG_North, IMG_South, IMG_West i IMG_East ja que son aquestes columnes les que es necessiten per crear el KML que serà enviat al client.

Un cop hem analitzat i dissenyat la base de dades del nostre projecte, cal escollir un gestor de base de dades per tal de crear-la físicament. Per la seva naturalesa de software lliure, es decideix implementar la base de dades en PostrgeSQL, un sistema de gestió de

bases de dades objecte-relacional que es distribueix sota la llicència BSD (*Berkeley Sofware Distribution*), corresponent a una família de les llicències de programari lliure.

La versió de PostgreSQL utilitzada és la 8.3.5 amb la eina PostgreSQL *pgAdmin III*, en la versió 1.8.4, que permet la interactuació amb la base de dades mitjançant una interfície d'usuari visual, còmode i molt intuïtiva. Sobre aquesta versió de PostgreSQL es va instal·lar el mòdul PostGIS, en la seva versió 1.3.5. Aquest mòdul, desenvolupat sota la filosofia de codi obert, permet a les bases de dades creades en PostgreSQL la possibilitat de fer servir objectes geogràfics en una base dades objecte-relacional, com és el cas. PostGIS, a més, afegeix suport per a l'ús d'eines especials per aquests tipus de dades geogràfiques.

Creant la base de dades sobre una instal·lació PostgreSQL amb suport per a dades geomètriques, podrem fer servir funcions pròpies dels sistemes SIG com ara el càlcul de distàncies entre punts o, fins i tot, crear codi KML de forma directa amb una consulta SQL. Cal remarcar, però, que en aquest projecte no fem servir aquestes funcions però sí veiem molt interessant deixar la base de dades preparada per a suportar-les per futures ampliacions en els requeriments del projecte.

3.2.5. L'script

En aquest apartat analitzem l'script que generarà codi KML que serà retornat al client. Tal i com em comentat anteriorment, aquest script és desenvolupat en el llenguatge d'scripting PHP. El codi complet de l'script el podem trobar al final d'aquest document, a l'apartat dedicar als annexos.

Hi ha 2 fets necessaris per tal de que un script proporcioni correctament codi KML a través d'una xarxa, com és el cas del nostre projecte. Quan es fa una crida des del client, és a dir, des del navegador Google Earth, cap al servidor, aquest ha de retornar una resposta HTTP de codi 200, que correspon a una resposta estàndard de petició HTTP

satisfactòria; el camp C*ontent-Type* d'aquesta resposta ha de ser de tipus *application/vnd.google-earth.kml+xml*, ja que aquest és el *MIME type* associat a un arxiu KML. El segon requisit necessari per al bon funcionament entre client Google Earth i el servidor és que el codi KML proporcionat estigui ben format, tal i com hem comentat en el capítol anterior.

Podem desglossar el codi en 3 parts diferenciades. La primera part de l'script s'encarrega de fer la connexió amb la base de dades amb la que l'script necessitarà interactuar. Aquesta base de dades la descriurem més endavant però ara cal saber que residirà en el propi servidor on està l'script. En cas que no es pugui establir aquesta connexió, l'script no continuarà i això derivarà en un error en el navegador Google Earth del client.

La següent part de l'script s'encarrega d'adquirir les variables que ha rebut via HTTP i guardar-les en variables. De fet, el que es fa és discriminar les 4 coordenades que rebem en una única variable composta anomenada BBOX creant 4 noves variables a l'script anomenades $west, $south, $east i $north. Seguidament, recupera el valor de l'altre variable que ha rebut via HTTP, CÀMERA, que conté el valor de l'alçada en metres de la vista del navegador i el guarda dins la variable $lookatRange.

La tercera i ultima part de l'script és la que duu a terme la consulta a la base de dades per tal d'obtenir la imatge que encaixa en les coordenades que ha rebut. Ja hem comentat anteriorment, que la base de dades conté dues taules. La taula IMG_1 conté les imatges de nivell 1, mentre que la taula IMG_4 conté les imatges de nivell 2, és a dir, les imatges corresponents als quatre quadrants de la imatge original. En aquest moment és on l'alçada llindar que diferenciava entre un nivell i l'altre entra en joc. Es decideix que aquesta alçada sigui de 150 metres. Per tant, quan la variable $lookatRange sigui major que 150, efectuarem la consulta sobre la taula IMG_1. En altre cas, la consulta es realitzarà sobre la taula IMG_4. La consulta, en ambdós casos, retorna un path a una imatge, que serà la nova imatge que es veurà en el navegador del client.

Finalment, es retorna el codi KML que mostra la imatge. Primerament, és important remarcar la presència de les capçaleres en el fitxer que l'script retorna. Aquestes

capçaleres indiquen al navegador Google Earth que el que enviem és un fitxer KML. L'script retorna el codi KML que veiem a continuació:

```
<GroundOverlay>
  <name>
    Monitoratge d'entorn dinàmic
  </name>
  <description>
    Planta de producció
  </description>
  <Icon>
    <href> %s </href>
  </Icon>
  <LatLonBox>
    <north>%.14f</north>
    <south>%.14f</south>
    <east>%.14f</east>
    <west>%.14f</west>
    <maxAltitude>1000</maxAltitude>
  </LatLonBox>
</GroundOverlay>
```

GroundOverlay és l'element que dibuixa una imatge sobre el terreny en el navegador Google Earth. Conté dos elements que defineixen aquesta imatge:

- Icon: és l'element que indica la ubicació física de la imatge, especificada en el subelement <href>. En el nostre cas, el contingut de l'element <href> serà una URL semblant a la que hem indicat en l'apartat anterior, quan descrivíem el codi del fitxer que té el client amb el Network Link. En aquest cas, necessitem que aquesta URL faci referència a una imatge ubicada en el servidor, com podem veure a continuació:

http://SERVIDOR/NOM_FITXER_IMATGE

El valor que troben en l'script, %s, indica que serà una cadena que es calcularà en temps d'execució. L'script farà la consulta a la base de dades i omplirà la URL completa per a la imatge en qüestió.

- LatLonBox: especifica les coordenades on s'ubicarà la imatge dins el navegador Google Earth, que seran les mateixes que l'script ha rebut des del navegador. Aquest element té quatre subelements corresponents a les 4 coordenades geogràfiques i un cinquè element anomenat `<maxAltitude>` que indica la altitud màxima en metres a la que es podrà desplaçar l'usuari. Es decideix limitar-ho a 1000 per tal de que l'usuari no arribi a veure la curvatura de la terra que mostra Google Earth quan s'arriba a certa altura en el visor del navegador.

Al final de l'script, alliberem la connexió amb la base de dades tot tancant la connexió establerta des de l'script.

3.2.6. Diagrama d'objectes

Ja hem descrit tots els components del nostre projecte, com interactuen entre ells i quins són els fluxos d'informació que transcorren durant la execució de la aplicació. A la figura 3.4 podem veure un diagrama complet del nostre projecte, amb aquests components i la comunicació entre ells:

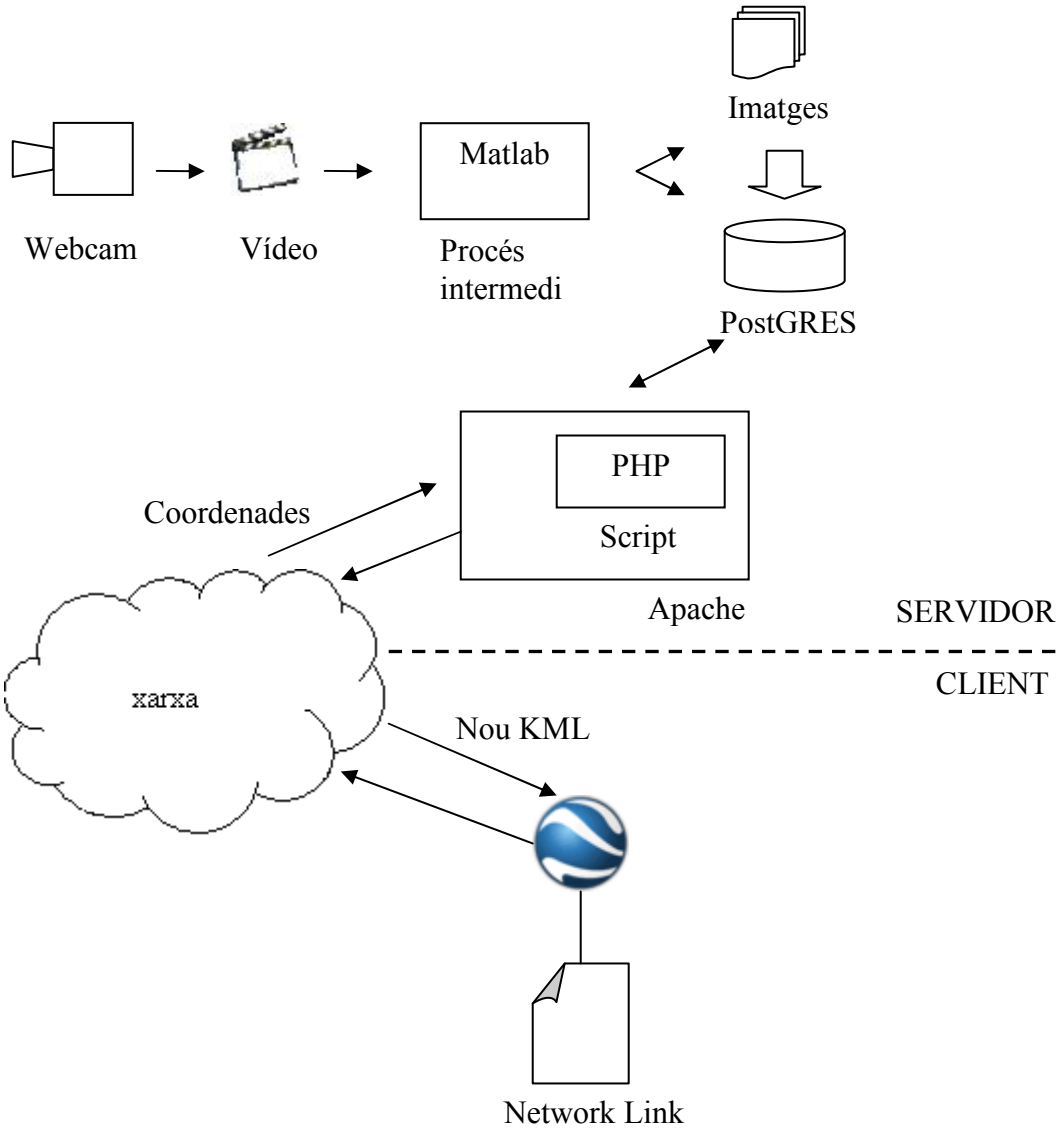

Figura 3.4

Capítol 4.

Implementacions avaluades

En aquest capítol repassarem les implementacions que s'han desenvolupat de la plataforma que hem descrit detalladament en el capítol anterior. La primera implementació consisteix en connectar directament els elements que hem comentat anteriorment, en una sola maquina. És a dir, tindrem un ordinador que actuarà com a client i com a servidor. Instal·larem, doncs, tots els elements que intervenen en el nostre sistema:

- Navegador Google Earth
- Entorn PHP i script
- Servidor web
- Base de dades PostgreSQL
- Matlab
- Webcam

Aquests elements es connectaran entre sí tal i com hem indicat anteriorment i s'avaluaran els resultats que s'hi obtenen pel que fa a la visualització des del navegador Google Earth. Veurem aquest procés en el punt 4.1.

Per tal d'intentar accelerar la visualització final en el navegador Google Earth, es va desenvolupar una segona implementació de l'arquitectura que es diferencia de l'anterior en el funcionament de la base de dades geogràfica. En aquesta segona versió, vam decidir ubicar la base de dades en memòria principal en comptes de memòria secundària per tal d'accelerar la transferència de les imatges. En el punt 4.2 expliquem per què es va decidir implementar aquesta millora i s'analitzen els resultats finals.

4.1. Implementació directa

Aquesta va ser la primera implementació que es va desenvolupar de l'arquitectura descrita durant el capítol 3. Es van connectar directament els diferents components en una mateixa màquina que va actuar tant de client com de servidor. El flux de processos seria el següent: la càmera capta les imatges d'un escenari, el procés Matlab converteix el vídeo de la càmera a imatges i les guarda a la base de dades. Mentre això succeeix, obrim un navegador Google Earth i hi carreguem el Network Link que es comunica amb l'script PHP, enviant les dades de les coordenades i alçada de la camera del visor via HTTP. Aquest script rep les dades, fa la consulta a la base de dades i recupera el camí a la imatge que s'hauria de mostrar en el navegador del client. Llavors, crea el codi KML per a mostrar aquesta imatge i la envia a l'aplicació Google Earth fent servir el protocol HTTP.

El primer problema que ens vam trobar va ser amb la propietat de refresc del Network Link del client. Aquesta propietat indica la freqüència amb la que el Network Link es recarregarà, és a dir, cada quan enviem la petició HTTP al servidor i, conseqüentment, cada quan es recarregarà la imatge en el navegador del client.

Ja hem explicat anteriorment que aquesta propietat s'indica en l'element anomenat <refreshInterval>, que és un subelement de <link>. Havíem dit que aquesta propietat admet un valor real, segons la documentació publicada per Google. Al realitzar les proves d'avaluació d'aquesta primera implementació descobrim que un refresc de 0,1 segons dona els mateixos resultats que un de 1 segon. Això voldria dir que la exploració d'un

escenari utilitzant aquesta tecnologia estaria limitada a un frame per segon. Per tant, semblaria poc aconsellable desenvolupar una aplicació per a entorns on es requerís una visualització a temps real on l'usuari hagués de realitzar un monitoratge de l'escenari amb rapidesa i fluïdesa en les imatges.

Després de realitzar vàries proves amb KML, provant varis sistemes de refresc, es descobreix una possibilitat d'augmentar la freqüència de refresc a valors per sota del segon. Es tracta d'enllaçar dos elements <NetworkLink> situats en dos fitxers KML diferents. De fet, només es tracta d'afegir un fitxer KML al nostre sistema. Aquest nou fitxer contindria un Network Link que apuntaria al mateix fitxer KML del client al que fins ara ens hem referit, tot seguint l'esquema que tenim a la figura 4.1.

D'aquesta manera, el client només haurà d'obrir el fitxer anomenat Network Link 1, que apuntarà al fitxer Network Link 2. Aquest serà el fitxer que apuntarà a l'script situat en el servidor. Així, el procés no ha canviat per a l'usuari, que només s'ha de preocupar d'obrir un fitxer, com fins ara havíem plantejat. És, doncs, important remarcar que per al client, la solució escollida per a resoldre el problema del refresc del Network Link és totalment transparent.

Cal remarcar que a la documentació de Google Earth no s'ofereix aquesta possibilitat d'obtenir taxes de refresc inferiors a 1 segon. És possible que, degut a la naturalesa de Google Earth, no s'hagi pensat en aquesta possibilitat. Recordem que Google Earth no està concebut per a entorns dinàmics, si no per a navegacions sobre entorns que no canvien constantment.

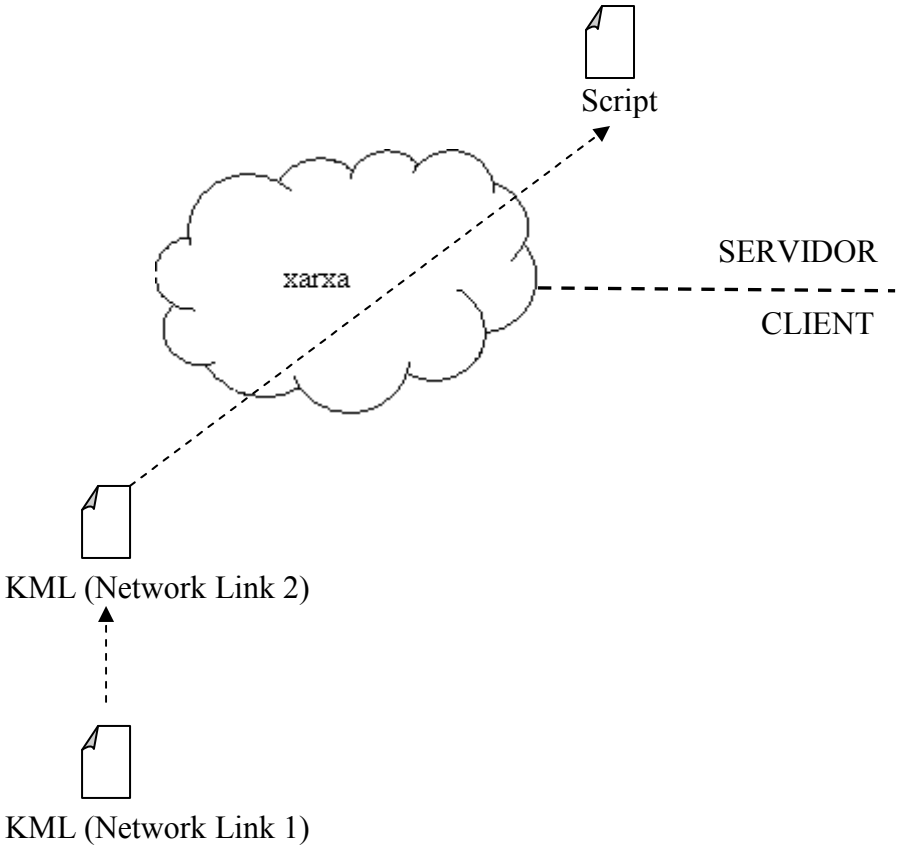

Figura 4.1

La taxa de refresc del segon Network Link la hem de fixar a un valor alt, ja que és el primer Network Link el que definirà la taxa de refresc amb la que es recarregaran les imatges en el navegador. De fet, l'efecte de fixar una taxa de refresc inferior a 1 en tots dos elements és força negatiu atès a les proves que hem fet, ja que es produeix un efecte de parpelleig molt elevat que fa impossible navegar per l'escenari de forma còmode. Deixem el refresc del segon element Network Link a 10 segons i variem el del primer Network Link per tal de veure quins resultats s'aconsegueixen. En realitat, el que estem fent és cridar l'script cada 10 segons i, a més, també el criden amb la freqüència que indiquem en el primer Network Link. Això no comporta cap efecte negatiu visual notable per a l'usuari.

Si provem de fixar la taxa del primer Network Link a 1 segon, obtenim una visualització força fluida pel que fa als canvis entre frames, tot i que es nota un petit interval entre dos frames consecutius en el que no es veu cap imatge. La velocitat és realment bona, ja que veiem les imatges gairebé a temps real. De fet, no ho veiem a temps real totalment per les prestacions de la càmera. Creiem que amb una camera de millors prestacions aconseguiríem una navegació en temps real.

No obstant, una freqüència d'un frame per segon sembla força baixa. Fixem una taxa de 2 frames per segon en el primer Network Link i veiem que continua existint un interval entre dos frames consecutius en els que no es veu cap imatge. Això provoca un efecte de parpelleig força molest que fa poc còmode la navegació amb Google Earth. Òbviament, a mesura que augmentem la freqüència de refresc, aquest efecte de parpelleig s'acusa més. Les proves amb una taxes de refresc de 10 frames per segon, és a dir, 0.1 segons d'interval de refresc donen uns resultats força negatius degut al efecte comentat.

Podríem dir que les primeres proves no donen uns resultats massa satisfactoris per a taxes de refresc superiors a 0.5 segons. Això podria ser degut a que les imatges no estan preparades a temps des del servidor quan són demanades per Google Earth.

4.2. Implementació amb base de dades sobre memòria

Degut als resultats de la primera implementació, que van semblar força millorables, es va decidir realitzar una segona implementació de l'aplicació amb l'objectiu de disminuir el temps de visualització entre dos frames consecutius en el navegador de Google Earth. Primerament s'havia d'analitzar quin podia ser el coll d'ampolla del sistema, l'element que provocava que les imatges no arribessin al visor del client en el temps esperat.

Es va pensar que possiblement el problema radicava en la transferència de les dades, sent les imatges la part a tenir més en compte ja que representen gairebé la totalitat de les dades que transferim. En aquest aspecte, hi hauria dues formes de millorar la rapidesa de

la transferència de les imatges. Hi ha dos trams on les imatges són transferides al llarg del sistema que hem dissenyat. El primer tram el trobem localment al servidor; estem treballant sobre una base de dades que guarda dades sobre imatges emmagatzemades en el propi servidor on resideix la base de dades. Quan l'script fa la consulta a la base de dades, es recupera un path que apunta a una imatge guardada en memòria secundària i es transfereix al client mitjançant la xarxa. Precisament aquest seria el segon tram on es transfereix la imatge, del servidor al client. Aquí la velocitat de transferència depèn del medi per on es transporta, de la congestió de la xarxa i altres elements físics que poden ser independents del nostre sistema.

Una forma de millorar la rapidesa amb la que la imatge és transferida seria ubicant la imatge en memòria principal en comptes de fer-ho en el disc. Això faria que la imatge estigués preparada per ser enviada al client força més aviat que quan la imatge s'ha d'anar a buscar a memòria secundària i podria derivar en una visualització més ràpida en el navegador. Es decideix, doncs, emmagatzemar les imatges directament en memòria principal sense passar per disc.

Per a dur a terme aquesta tasca es necessita una aplicació en el servidor que permeti al sistema operatiu treballar sobre memòria principal com si es tractés d'un disc. Es decideix utilitzar l'eina *VirtualDrive Pro*, en la seva versió 11.0, per tal de dur aquesta tasca degut a la seva simplicitat. Amb aquesta eina, aconseguim que el sistema operatiu assigni una nova unitat a una porció de la memòria principal per utilitzar-la com si fos una unitat d'emmagatzematge. Així doncs, només cal modificar el procés Matlab per tal de que guardi les imatges sobre aquesta unitat. Caldrà, també, indicar aquest canvi en el camp IMG_Path de les imatges noves a la base de dades. També caldrà indicar a l'Apache un nou directori on hi hauran els documents que pot servir a l'exterior, que serà la nova unitat creada i que s'indica en el fitxer de configuració httpd.conf.

Tanmateix, amb aquesta implementació ens apareix un nou problema a tenir en compte. Ara les imatges amb les que treballa el nostre projecte resideixen en la memòria principal del servidor. Això implica un alt grau d'ús de la memòria principal, és a dir, hi

necessitem molt espai. Fins ara guardàvem les imatges a disc i no teníem en compte la capacitat del mateix. Només necessitaríem moure els fitxers que contenen les imatges per problemes de capacitat a llarg termini i és un fet que no hem considerat de rellevància en aquest projecte. No obstant, si treballem en memòria principal sí necessitarem algun mecanisme per a moure les imatges de memòria a disc a curt termini ja que el cost econòmic que suposaria tenir una memòria principal amb molta capacitat podria ser molt alt. De fet, tampoc necessitem que les imatges resideixin en memòria un cop ja han sigut mostrades al client. Només caldria conservar aquestes imatges per possibles consultes posteriors, és a dir, constituirien un històric d'imatges. Aquest conjunt d'imatges haurien, doncs, de ser guardades a disc.

Així, es decideix dissenyar un mecanisme que permeti moure les imatges de memòria principal a disc cada cert temps per tal de no ocupar la totalitat de l'espai en memòria i disposar d'un històric d'imatges. Es pensen dues possibilitats per a dur a terme aquesta tasca. A continuació descrivim aquestes dues possibilitats, com s'haurien d'implementar i quines serien les avantatges i els inconvenients que presentarien.

La primera solució constaria d'un procés que s'executaria cada cert temps i que realitzaria el traspàs físic dels fitxers de les imatges de memòria principal cap a memòria secundària. També hi hauria un segon element que efectuaria el canvi en la base de dades, és a dir, actualitzaria el camp IMG_Path de les taules IMG_1 i IMG_4 que conté la imatge del camí físic on resideix la imatge. Aquests dos processos anirien sincronitzats de forma que un cop s'ha acabat el traspàs físic de les imatges es procediria a fer els canvis a la base de dades. Com que el nom del fitxer de la imatge conté el seu identificador a la base de dades, és a dir, el valor del camp IMG_Id, no hi hauria problema en saber quin registre de la base de dades correspon a un fitxer donat.

Per tal de decidir quines imatges seran traspassades i quines no, es fixarà a priori un valor que correspondrà al nombre d'imatges que es mouran cada cop que aquests processos es disparin. Aquest valor dependrà de la capacitat que tingui la memòria principal del servidor, i serà directament proporcional a aquesta capacitat, és a dir, a més capacitat més

imatges s'hauran de moure cada cop ja que hi caben més. De fet, a més capacitat menys necessitat hi ha de fer el traspàs físic i, per tant, aquests processos es dispararan amb menys freqüència i, per tant, hi hauran més imatges a moure.

La segona solució consistiria en guardar les imatges al disc en el moment en que també són guardades en memòria. Així, cada cop que el procés Matlab que hem descrit en capítols anteriors guardi una imatge en memòria, caldrà que també guardi el mateix fitxer a disc. D'aquesta manera, només caldrà un procés que esborri les imatges de memòria i no faci cap traspàs físic de fitxers. També caldrà el procés que actualitzi la base de dades per tal de canviar el camp IMG_Path de les taules IMG_1 i IMG_4 tal i com hem explicat en l'anterior cas.

Aquestes dues solucions estan basades en dues tècniques d'escriptura i sincronització entre memòria caché i memòria principal que en el nostre cas apliquem en la sincronització entre memòria principal i secundària. La primera la podríem comparar amb la tècnica anomenada *write-through*, que implica actualitzar els canvis en memòria principal en el moment en que es graven en memòria caché. La segona opció seria comparable a la tècnica *write-back*, que no actualitza les dades en memòria principal en el mateix moment en que es fa la escriptura en memòria caché si no que marca les dades no sincronitzades en memòria principal com a pendents i les actualitza a posteriori. En el nostre cas, no caldria fer aquesta diferenciació ja que les imatges pendents de ser traspassades no existeixen en memòria secundària.

Per a les proves que s'han realitzat per aquesta segona implementació no s'ha vist necessari realitzar els processos d'històrics, tot i que era necessari analitzar i dissenyar algun mecanisme per fer-ho, veient que és viable utilitzar la memòria principal per a guardar-hi les imatges sense tenir problemes d'espai a curt termini. Creem, doncs, una unitat d'emmagatzematge dins la memòria RAM amb l'ajuda de la eina *VirtualDrivePro* de 300 Mb, ja que es disposa d'una memòria principal de 2 Gb. A més, modifiquem el procés Matlab per a que treballi en aquesta unitat i li indiquem a l'Apache el nou directori, que serà directament la nova unitat.

Tal i com hem fet per avaluar la primera implementació, s'han provat diferents valors per a la taxa de refresc del primer Network Link. Així, per a una taxa de refresc d'1 segon, s'obté una navegació una mica més fluïda que en la primera implementació, experimentant positivament el fet de que es necessita més temps per a que les imatges estiguin llestes per ser enviades al client.

Provant una freqüència de 10 frames per segon, en la primera implementació obteníem uns resultats no gaire bons degut a l'efecte de parpelleig. En aquesta segona implementació, aconseguim uns resultats força similars. Provant freqüències més altes, ens trobem que és impossible navegar per l'escena ja que aquest efecte de parpelleig és massa acusat.

Capítol 5.

Conclusions

En aquest capítol revisarem quins eren els objectius i les tasques que volíem dur a terme en aquest projecte. Presentarem els resultats obtinguts i es discutirà si s'han assolit els objectius que s'havien proposat inicialment. Finalment, veurem quines són les possibles línies de continuació que es podrien seguir a partit de les conclusions extretes amb el projecte.

5.1. Motivacions i objectius

Un cop hem finalitzat el projecte, revisarem els objectius i les tasques que s'han dut a terme al llarg del desenvolupament del mateix. Recordem que l'objectiu principal era avaluar quin era el rendiment que ofereix Google Earth com a navegador geogràfic en entorns dinàmics. Per a complir aquest objectiu, necessitàvem desenvolupar una aplicació de base que es comuniqués amb la aplicació Google Earth per tal d'oferir un monitoratge d'un escenari gravat per una camera. Volíem que aquest monitoratge resultés el més senzill i fluid possible i decidir si és possible arribar a una freqüència de visualització de 25 frames per segon, és a dir, el màxim que pot processar l'ull humà.

Ara, podem afirmar que és totalment viable realitzar una aplicació que permeti a un usuari visualitzar un entorn dinàmic mitjançant Google Earth. Hem desenvolupat dues implementacions diferents d'una aplicació base sobre la que hem avaluat el rendiment de Google Earth per al nostre problema. Hem vist que aquesta tecnologia permet treballar amb entorns dinàmics, a diferència del que ofereix normalment als usuaris. Avui dia, molts usuaris a Internet utilitzen Google Earth per a explorar el món, amb unes imatges emmagatzemades en els servidors de Google que són actualitzades cada cert temps, que

poden ser mesos o anys. Tanmateix, hem demostrat que és possible fer servir aquesta tecnologia per a entorns on es requereixen actualitzacions constants.

No obstant, també hem vist que el rendiment de Google Earth a l'hora d'obtenir taxes de refresc d'imatges en el navegador per sota dels 10 frames per segon és molt pobre degut als constants canvis i efectes de parpelleig. És possible que aplicant tècniques que apuntem en el següent punt, en les línies de continuació, s'aconsegueixin millors resultats i s'elimini, en part, aquest efecte de parpelleig. No obstant, creiem que serà difícil aconseguir unes taxes de refresc molt altes en una aplicació pensada per entorns que no canvien. A mes, ja vam dir en el primer capítol, que el fet d'escollir Google Earth com a tecnologia sobre la qual desenvolupar el nostre projecte implicaria un *overhead* al tractar-se d'un model d'aplicacions que interactuen amb aplicacions. En el nostre cas, hem desenvolupat una aplicació base sobre la que treballava l'aplicació Google Earth.

5.2. Resultats

Tenint en compte els resultats que hem aconseguit, podríem dir que seria possible utilitzar la tecnologia Google Earth per a entorns dinàmics on la exploració de l'entorn no requereixi d'una visualització amb una alta taxa de refresc de les imatges.

Avaluant Google Earth i la segona implementació aplicació base que hem desenvolupat, que és la que dona millors resultats, arribem a una visualització de 10 frames per segon amb un efecte de parpelleig que és força elevat i fa impossible que l'usuari pugui dur a terme un monitoratge de forma fluida i còmode. Si treballem amb 5 frames per segon, aconseguim reduir força aquest efecte negatiu. Així, podem concloure que per a entorns on no es requereixi una taxa de refresc superior a 5 frames per segon, Google Earth és una tecnologia totalment vàlida per a la qual desenvolupar aplicacions del tipus que hem tractat en aquest projecte.

Es possible que aplicant algunes tècniques per accelerar la transferència de les imatges, i que discutirem en el punt 5.3, s'aconsegueixi reduir l'efecte de parpelleig en el navegador. No obstant, pensem que aquest efecte hi serà sempre present ja que Google Earth necessita recarregar cada cop un codi KML i, per tant, necessita un temps per tal d'interpretar el codi que rep i mostrar els resultats en el visor. Tal i com hem comentat al llarg del projecte, creiem que Google Earth ha estat pensat per altres tipus d'entorns i no per a dur a terme un monitoratge on es requereix una actualització constant, tal com 10 frames per segon.

5.3. Línies de continuació

Un cop hem acabat el projecte, se'ns plantegen vàries maneres d'aconseguir millors resultats en la visualització d'entorns dinàmics amb Google Earth. Hi hauria vàries tècniques que podríem aplicar sobre la aplicació de base que hem desenvolupat al llarg d'aquest projecte i que creiem que ajudarien a obtenir millors resultats. A continuació enumerem i expliquem aquestes tècniques que podrien ser aplicades en futures línies de continuació.

5.3.1. Diferències d'imatges

Ja hem comentat varis cops en aquest projecte que els entorns que volem explorar amb Google Earth tenen un alt nivell de dinamisme, és a dir, varien constantment en el temps. No obstant, és clar que entre dos frames consecutius del vídeo que estem enregistrant, els canvis seran mínims. Al llarg d'aquest projecte hem plantejat els exemples d'una planta de producció o d'una zona d'activitats logístiques; en ambdós casos, dos frames consecutius del vídeo que capta la camera variaran únicament en la posició d'algun robot o d'algun cotxe o operari. No obstant, la resta de la imatge serà molt possiblement idèntica a l'anterior. Aquest fet, que s'aprofita en tècniques de compressió de vídeo,

podria utilitzar-se també en el nostre projecte per tal d'obtenir millors resultats en la aplicació final.

Per aprofitar aquest fet, caldria modificar la filosofia amb la que hem treballat a l'hora de treballar amb les imatges que capta la camera. D'alguna manera, hauríem de deixar de treballar amb imatges pròpiament dites per passar a treballar amb diferències d'imatges, és a dir, en un instant i no tenim una imatge si no la variació que hi ha hagut en l'instant i respecte l'instant i – 1. Com que ja hem raonat que aquesta variació seria mínima, tindríem molta menys informació per a descriure l'instant i. Òbviament, treballar amb menys informació comportaria més velocitat en la aplicació final, menys requeriments de capacitat de memòria i, en definitiva, una visualització més fluida ja que entre el servidor i el client transferiríem molta menys quantitat d'informació.

5.3.2. Regions d'imatges

Una tècnica que podríem aplicar, i que va lligada amb l'anterior punt, seria treballar amb regions de les imatges. Es tractaria de dividir les imatges de forma quadricular, per exemple. Cada quadrícula seria una regió de la imatge. Així, en comptes de transferir cada cop una imatge sencera entre el servidor i el client, transferiríem una o vàries regions de la imatge: aquelles que haguessin sofert una variació respecte l'anterior frame.

Hauríem, doncs, de comunicar d'alguna forma a l'aplicació Google Earth quines són les regions que necessiten actualitzar-se, tot transferint-li aquestes regions en un o varis fitxers jpg. Creiem que aquesta tècnica seria molt interessant de ser aplicada en el nostre context. Caldria un estudi previ de l'escenari a desenvolupar per tal de decidir en quantes regions s'hauria de dividir un frame del vídeo per aconseguir els millors resultats en la visualització final amb Google Earth.

5.3.3. Procés Matlab

Una fet creiem que pot disminuir el rendiment del nostre projecte és el procés Matlab que hem descrit en el capítol 3. Aquest procés s'encarregava de convertir el vídeo que capta la camera a imatges i emmagatzemar-les en la base de dades de la forma que hem detallat. Sabem, però, que Matlab és un entorn de programació molt versàtil per a treballar amb imatges però que no presenta una velocitat de processament molt alta, sobretot comparat amb llenguatges de programació de més baix nivell com pot ser C. Creiem que podríem aconseguir més velocitat de processament si en comptes de fer ser l'entorn Matlab creéssim una petita aplicació codificada en llenguatge C, per exemple, que fes la mateixa tasca. Aquesta aplicació estaria sempre executant-se en el servidor, tal i com fa el procés Matlab. S'hauria d'avaluar fins a quin punt aquest canvi afectaria als resultats finals de visualització en el navegador, però estem segurs que el processament i emmagatzematge d'imatges en el servidor es faria d'una forma més eficient i més ràpida.

Referències

1. Introducció al KML a Google
http://code.google.com/intl/ca/apis/kml/documentation/
Última data consulta: 13/03/2009

2. Documentació KML a Google
http://code.google.com/intl/ca/apis/kml/documentation/kmlreference.html
Última data consulta: 26/08/2009

2. Documentació de la toolbox *Image Acquisition* per a Matlab
http://www.mathworks.com/matlabcentral/fileexchange/22792
Última data consulta: 15/04/2009

3. Introducció al processament d'imatges amb Matlab
http://www.cogs.susx.ac.uk/users/davidy/compvis/matlab_demos/intro_demo.html
Última data consulta: 15/04/2009

4. Documentació distribució Apache XAMPP
http://www.apachefriends.org/en/xampp.html
Última data consulta: 07/06/2009

5. Generación simple de KML con PHP y PostGIS
http://neogeografia.wordpress.com/2007/08/12/generacion-simple-de-kml-con-php-y-postgis
Última data consulta: 16/02/2009

6. Generación de salidas dinámicas simples en KML
http://pcsig2007.wikispaces.com/Generaci%C3%B3n+de+salidas+din%C3%A1micas+simples+en+KML+a+partir+de+Consultas+Espaciales+creadas+con+SFQL
Última data consulta: 20/02/2009

7. The KML Screen Overlay Maker Utility
http://freegeographytools.com/2007/the-kml-screen-overlay-maker-utility
Última data consulta: 23/01/2009

8. Documentació de PostgreSQL v8.3
http://www.postgresql.org/docs/8.3/static/index.html
Última data consulta: 15/06/2009

9. Aplicacions dinàmiques a Google Earth amb Network Links
http://geochalkboard.wordpress.com/2007/09/04/dynamic-google-earth-applications-with-network-links/
Última data consulta: 07/03/2009

10. Google Earth Network Link
http://www.gearthblog.com/blog/archives/2007/04/the_google_earth_net.html
Última data consulta: 01/03/2009

11. Grups Google de suport per desenvolupadors de codi KML
http://groups.google.es/group/kml-support
Última data consulta: 26/05/2009

12. Documentació llenguatge PHP
http://www.php.net/
Última data consulta: 18/06/2009

13. Tutorial codi KML per Mano Marks i Brian Hamlin
http://www.youtube.com/watch?v=QzS_shIzfcM
Última data consulta: 23/03/2009

14. “Using Google Earth to Access and Display Emissions Data”
David Mintz (U.S. Environmental Protection Agency)
(Maig 2007)

Apèndix 1.

Codi Procés Matlab

```
% Precisió de 14 decimals per als nombres Reals
format long g;
north= 0.00066986242119;
south=-0.00066986242119;
east=  0.00077883890155;
west= -0.00077883890155;

% Servidor de base de dades
host = 'localhost';

% Usuari i contrasenya de la base de dades
user = 'postgres';
password = '1107Aa25';

% Nom de la base de dades
dbName = 'PostgreSQL30';


% Creacio de la connexio
dbConn = database(dbName, user , password)%, jdbcDriver,
jdbcString);


%%%%%%%%%%%%%%%
% Taula IMG_1 %
%%%%%%%%%%%%%%%

sql='select "IMG_Id" from "IMG_1"';
curs = exec(dbConn, sql);
curs = fetch(curs);

%Si no hi ha elements, el proper Id es 1, si no, el proper
es el maxim+1
```

```
if curs.Data{1} == 'No Data'
    maxId_1 = 1;
else
    maxId_1 = max(curs.Data{1:end}) + 1;
end


%%%%%%%%%%%%%%%
% Taula IMG_4 %
%%%%%%%%%%%%%%%

sql='select "IMG_Id" from "IMG_4"';
curs = exec(dbConn, sql);
curs = fetch(curs);

%Si no hi ha elements, el proper Id es 1, si no, el proper
es el maxim+1
if curs.Data{1} == 'No Data'
    maxId_4 = 1;
else
    maxId_4 = max(curs.Data{1:end}) + 1;
end


% Creacio de l'objecte que recull el video. Fixem
resolucio.
vid = videoinput('winvideo',1, 'RGB24_320x240');
set(vid,'TriggerRepeat',Inf);
vid.FrameGrabInterval = 1;



% Comencem a adquirir frames
start(vid)

% Perdrem el primer frame per agafar mides
frame = getdata(vid,1);
[h,w,c]=size(frame);

%El directori es la unitat K: (Memoria Primaria)
directory = ['K:\\'];
```

```
while(1)

    fileName = ['a-' num2str(maxId_1) '.bmp']

    fileName1 = ['a11-' num2str(maxId_4) '.bmp'];
    fileName2 = ['a12-' num2str(maxId_4) '.bmp'];
    fileName3 = ['a13-' num2str(maxId_4) '.bmp'];
    fileName4 = ['a14-' num2str(maxId_4) '.bmp'];

    %Seguent frame
    frame = getdata(vid,1);

    %Disminuim resolucio i guardem
    frameTemp = IMRESIZE(frame,0.5,'nearest');
    imwrite(uint8(frameTemp), [directory fileName], 'bmp');

    %Retallem imatge i guardem
    imwrite(frame(1:h/2,1:w/2,:),[directory fileName1],
    'bmp');
    imwrite(frame(1:h/2,w/2+1:end,:),[directory fileName2],
    'bmp');
    imwrite(frame(h/2+1:end,1:w/2,:),[directory fileName3],
    'bmp');
    imwrite(frame(h/2+1:end,w/2+1:end,:),[directory
    fileName4], 'bmp');

    %Insercions a la base de dades
    sql= ['insert into "IMG_1" values (' num2str(maxId_1)
     ', ''' fileName  ''', ' num2str(north) ','
     num2str(south) ',' num2str(west) ',' num2str(east)
     ')'];
    curs = exec(dbConn, sql);

    sql= ['insert into "IMG_4" values (' num2str(maxId_4)
     ', ''' fileName1  ''', ' num2str(north) ',' num2str(0)
     ',' num2str(west) ',' num2str(0) ',' num2str(1) ')'];
    curs = exec(dbConn, sql);

    sql= ['insert into "IMG_4" values (' num2str(maxId_4)
     ', ''' fileName2  ''', ' num2str(north) ',' num2str(0)
     ',' num2str(0) ',' num2str(east) ',' num2str(2) ')'];
    curs = exec(dbConn, sql);

    sql= ['insert into "IMG_4" values (' num2str(maxId_4)
     ', ''' fileName3  ''', ' num2str(0) ',' num2str(south)
     ',' num2str(west) ',' num2str(0) ',' num2str(3) ')'];
```

```
    curs = exec(dbConn, sql);

    sql= ['insert into "IMG_4" values (' num2str(maxId_4)
     ', ''' fileName4  ''',' num2str(0) ',' num2str(south)
     ',' num2str(0) ',' num2str(east) ',' num2str(4) ')'];
    curs = exec(dbConn, sql);

    %Seguents valors per als identificadors
    maxId_1 = maxId_1+1;
    maxId_4 = maxId_4+1;

end

%Aturem video
stop(vid)
```

Apèndix 2.

Codi Script PHP

```
<?
require("config.php");

/* Connexio amb la base de dades */

$conn = pg_connect("host=localhost
   port=5432
   dbname=postgis
   user=postgres
   password=aA9623B");

if (!$conn): ?>
    <H1>Failed connecting to postgres database</H1> <?
    exit;
endif;

/* Recuperem les variables que ens ha enviat el
navegador vía HTTP */

$bbox = split('[,]', $_GET['BBOX']);
$camera = split('[,]', $_GET['CAMERA']);

$west = (float) $bbox[0];
$south = (float) $bbox[1];
$east = (float) $bbox[2];
$north = (float) $bbox[3];

$center_lng = (($east - $west) / 2) + $west;
$center_lat = (($north - $south) / 2) + $south;


$lookatRange = (float) $camera[2];
```

```
/* Si estem a més de 150 metres d'alçada
veurem una imatge de la taula IMG_1 */

if ($lookatRange>150)
{

     /*  Consulta a la base de dades  */
     $sql = 'SELECT * FROM "IMG_1" WHERE "IMG_Id" = (SELECT
     MAX("IMG_Id") FROM "IMG_1")';

     $qu = pg_exec ($conn, $sql);
     $data = pg_fetch_object($qu);

     /* Obtenim cami de la imatge i coordenades  */
     $path1 = "http://localhost/" . $data->IMG_Path;
     $north1 = $data->IMG_North;
     $south1 = $data->IMG_South;
     $west1 = $data->IMG_West;
     $east1 = $data->IMG_East;

     /* Escrivim KML */
     Header('Content-Type: application/vnd.google-
     earth.kml+xml\n');

     printf('<?xml version="1.0" encoding="UTF-8"?><kml
     xmlns="http://www.opengis.net/kml/2.2"><Folder>');

     printf('
     <GroundOverlay>
      <name>Monitoratge d'un d'entorn dinàmic </name>

      <Icon>
        <href>%s</href>
      </Icon>

       <LatLonBox>
        <north>%.14f</north>
        <south>%.14f</south>
        <east>%.14f</east>
        <west>%.14f</west>
       </LatLonBox>

    </GroundOverlay></Folder></kml>
     ',$path1, $north1, $south1, $east1, $west1);
```

```
}


/* Si no,
veurem quatre imatges de la taula IMG_4 */

else
{
     /*  Consulta a la base de dades  */
     $sql = 'SELECT * FROM "IMG_4" WHERE "IMG_Id" = (SELECT
     MAX("IMG_Id") FROM "IMG_4")';

     $qu = pg_exec ($conn, $sql);

     /* Obtenim cami de la imatge i coordenades  */
     $data = pg_fetch_object($qu);
     $path1 = "http://localhost/" . $data->IMG_Path;
     $north1 = $data->IMG_North;
     $south1 = $data->IMG_South;
     $west1 = $data->IMG_West;
     $east1 = $data->IMG_East;

     /* Obtenim cami de la imatge i coordenades  */
     $data = pg_fetch_object($qu);
     $path2 = "http://localhost/" . $data->IMG_Path;
     $north2 = $data->IMG_North;
     $south2 = $data->IMG_South;
     $west2 = $data->IMG_West;
     $east2 = $data->IMG_East;

     /* Obtenim cami de la imatge i coordenades  */
     $data = pg_fetch_object($qu);
     $path3 = "http://localhost/" . $data->IMG_Path;
     $north3 = $data->IMG_North;
     $south3 = $data->IMG_South;
     $west3 = $data->IMG_West;
     $east3 = $data->IMG_East;

     /* Obtenim cami de la imatge i coordenades  */
     $data = pg_fetch_object($qu);
     $path4 = "http://localhost/" . $data->IMG_Path;
     $north4 = $data->IMG_North;
     $south4 = $data->IMG_South;
     $west4 = $data->IMG_West;
     $east4 = $data->IMG_East;
```

```
 Header('Content-Type: application/vnd.google-
 earth.kml+xml\n');

 printf('<?xml version="1.0" encoding="UTF-8"?><kml
 xmlns="http://www.opengis.net/kml/2.2"><Folder>');

 printf('
 <GroundOverlay>
  <name>Monitoratge d'un d'entorn dinàmic </name>
  <Icon>
    <href>%s</href>
  </Icon>
   <LatLonBox>
    <north>%.14f</north>
    <south>%.14f</south>
    <east>%.14f</east>
    <west>%.14f</west>
  </LatLonBox>
</GroundOverlay>

 <GroundOverlay>
  <name>Monitoratge d'un d'entorn dinàmic 2</name>
  <Icon>
    <href>%s</href>
  </Icon>
   <LatLonBox>
    <north>%.14f</north>
    <south>%.14f</south>
    <east>%.14f</east>
    <west>%.14f</west>
  </LatLonBox>

</GroundOverlay>
 <GroundOverlay>
  <name>Monitoratge d'un d'entorn dinàmic 3</name>
  <Icon>
    <href>%s</href>
  </Icon>
   <LatLonBox>
    <north>%.14f</north>
    <south>%.14f</south>
    <east>%.14f</east>
    <west>%.14f</west>
  </LatLonBox>
</GroundOverlay>
```

```
      <GroundOverlay>
       <name>Monitoratge d'un d'entorn dinàmic 4</name>
       <Icon>
         <href>%s</href>
       </Icon>
        <LatLonBox>
         <north>%.14f</north>
         <south>%.14f</south>
         <east>%.14f</east>
         <west>%.14f</west>
       </LatLonBox>
     </GroundOverlay>
      </Folder></kml>
      ',$path1, $north1, $south1, $east1, $west1,
       $path2, $north2, $south2, $east2, $west2,
       $path3, $north3, $south3, $east3, $west3,
       $path4, $north4, $south4, $east4, $west4);

}

/* Tanquem connexio amb base de dades */
pg_free_result($qu);
pg_close($conn);

?>
```