Títol: Implementació de l'algorisme ROAM

Volum: 1/1
Alumne: Albert Astals Cid

Director/Ponent: Lluís Pérez Vidal
Departament: LSI
Data: 8 de juny de 2005

DADES DEL PROJECTE

Títol del Projecte: Implementació de l'algorisme ROAM

Nom de l'estudiant: Albert Astals Cid

Titulació: Enginyeria Informàtica

Crèdits: 37.5

Director/Ponent: Lluís Pérez Vidal

Departament: LSI

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President:

Vocal:

Secretari:

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índex

1 Com usar aquesta memòria

Aquest document està estructurat en dotze seccions bàsiques. El document segueix una forma lògica i seqüencial d'explicar els conceptes de forma que no calgui anar endavant i enrera per entendre les idees, per aquesta raó pot semblar que en algun moment s'estigui repetint alguna idea, però serà perquè s'està ampliant l'explicació d'algun concepte. Les seccions de la memòria són:

1. *Com usar aquesta memòria* - Explica quins continguts es poden trobar a cadascuna de les seccions per tal que el lector pugui decidir si pot saltar-se'n alguna, ja que els conceptes dels quals es parlen ja li són coneguts, així com veure quines seccions expliquen coses que ell desconeix, i per tant ha d'estar més concentrat en la seva lectura.

2. *Objectius i motivacions* - En aquesta secció es parla dels objectius que es volen assolir amb la realització d'aquest projecte final de carrera així com de les motivacions que han dut a l'alumne a realitzar-lo.

3. *Introducció* - Aquest capítol introdueix el concepte bàsic en el que es basa el projecte, l'algorisme ROAM, així com un petita explicació de perquè és necessari tenir algorismes per la representació de terrenys de forma dinàmica i l'explicació dels dos formats de fitxers de dades acceptats, DEM i mapes simples.

4. *Especificació* - Aquesta part de la memòria explica tot allò relacionat amb l'especificació: el model conceptual, el model de casos d'ús, els requeriments funcionals i no funcionals, i, una petita descripció informal de cada classe.

5. *Disseny* - Explica el pas de l'especificació al disseny, hi podem trobar un model conceptual modificat per acomodar-se als requeriments juntament amb els diagrames de seqüència.

6. *Implementació* - Aquesta part de la memòria parla de tot allò relacionat amb la programació del projecte però que no forma part ni de l'especificació ni del disseny, com per exemple l'explicació sobre l'eina escollida per la generació automàtica de Makefiles.

7. *Captures de pantalla* - En aquesta secció es poden trobar captures de pantalla de l'aplicació en funcionament per poder fer-se una idea de com funciona.

8. *Recursos i Anàlisi de costos* - En aquesta part de la memòria es parla de tot allò que ha fet falta per realitzar el projecte, la seva planificació i l'anàlisi de quan hauria costat desenvolupar aquest projecte en cas que s'haguès tractat d'un projecte comercial.

9. *Conclusions* - Explica que s'ha après en la realització d'aquest projecte així com possibles modificacions futures que es podrien dur a terme.

10. *Bibliografia* - Relació de totes les fonts consultades en la realització del projecte.

11. *Índex de figures* - Llistat de totes les figures que es troben a la memòria, de forma que en cas que el lector vulgui localitzar-ne una pugui fer-ho ràpidament.

12. *Annex* - Explicacions sobre temes no estrictament relacionats amb el projecte que poden ser d'ajuda per acabar d'entendre algunes parts de la memòria.

2 Objectius i motivacions

Els continguts relacionats amb els objectius que es volen assolir amb aquest projecte són una sintetització dels requisits expressats a la secció 4.

L'objectiu principal d'aquest projecte és la implementació de l'algorisme ROAM. Per aconseguir-ho tant el projectista com el director del projecte han decidit que la realització del projecte acabarà materialitzant-se en una aplicació amb una llicència de programari lliure. Aquesta aplicació implementarà l'algorisme ROAM així com lectors de diferents formats de fitxers per obtenir dades que puguin fer funcionar l'algorisme i una interfície simple per tal que l'usuari pugui controlar l'aplicació amb facilitat.

Com a objectius secundaris, es vol que l'aplicació sigui el més portable possible. La portabilitat es vol a dos nivells, el primer dels nivells és la indepèndencia del sistema operatiu, el segon nivell de portabilitat que es desitja és que la interfície gràfica utilitzada per interactuar amb el nucli de l'aplicació que implementa l'algorisme ROAM es pugui canviar sense problemes.

Les motivacions per realitzar el projecte són variades, però es poden resumir en els següents punts:

- El món dels gràfics tridimensionals generats per computador és un món en expansió. En aquests moments genera una quantitat de beneficis superior a la del cinema, a més a més, hi ha gran demanda d'aplicacions de visualització tridimensional, especialment per a temes mèdics, però també en altres camps. Per tant, la realització d'un projecte de final de carrera en aquest camp ajuda a obtenir coneixements sobre aquesta àrea amb tanta projecció que molt possiblement siguin útils en el futur.

- El món dels gràfics per computador sempre m'ha interessat de manera especial. Per aquesta raó he cursat quasi totes les asignatures que s'imparteixen a la FIB relacionades amb aquest tema, per tant, aquest camp va ser el primer en el que vaig començar a buscar projectes de final de carrera. Un altre camp que també m'interessa, i del qual també vaig mirar els projectes disponibles, és el del món del programari lliure.

- Un cop analitzats tots els projectes que els professors vinculats a la FIB oferien relacionats amb el món dels gràfics per computador i del programari lliure, vaig anar a parlar amb en Lluís que em va explicar en profunditat de que tractava aquest projecte així com també hem va comunicar que la seva idea era que es desenvolupés com a programari lliure. Això significava que aquest projecte conjugava els dos camps que m'interessaven, per tant, vaig decidir que si en Lluís estava d'acord en que jo el portés a terme, el projecte de la implementació de l'algorisme ROAM era el projecte que jo desitjava fer.

3 Introducció

3.1 Representació interactiva de terrenys en 3D

La representació 3D de terrenys pot ser trivial o extremament complicada. Hi ha un primer grup d'aplicacions en que el terra és molt poc important i amb un polígon texturat i pla quasi bé ja n'hi ha prou, la majoria de jocs 3D entren dins d'aquest conjunt d'aplicacions, jocs de cotxes, arcades 3D, etc. En canvi hi ha un conjunt d'aplicacions on el terreny és part molt important de la visualització, dins d'aquest grup podem trobar els jocs de simulació, sobretot els d'avions, però també jocs de simulació tàctica on el terreny pot permetre amagar-nos dels enemics, etc., evidentment totes les aplicacions que tinguin alguna cosa a veure amb els mapes també tenen els terrenys com a part important.

A les aplicacions del primer grup, usar un algorisme especial per representar el terreny no té cap sentit, ja que aquest està format per pocs polígons i no cal fer cap optimització per representar-lo. En canvi les aplicacions on l'orografia té un paper important, la visualització del terreny passa a ocupar gran part dels cicles de rellotge que gastarà la nostra aplicació i, per tant, un algorisme optimitzat és necessari. A més a més, en la majoria dels casos no serà únicament necessari per optimitzar l'aplicació, sinó que serà completament necessari per tal que l'aplicació pugui funcionar, ja que si volguéssim representar tot el terreny faria falta un nombre ingent de triangles. Per exemple, si fem servir un esquema de representació com el de la figura 1, què és bastant dolent (semblaria que el terreny tingués forma de serreta), però que crea un triangle entre cada punt i els seus dos més propers, és a dir, el més exacte possible, per un terreny quadrat de NxN punts, caldrien $2 * (N-1)^2$, cosa que significa que per un terreny de $1000x1000$ punts (res exagerat), caldrien uns dos milions de triangles, una quantitat massa elevada.

Degut a aquesta necessitat d'optimització sorgeixen els algorismes de representació dinàmica de superfícies. Aquests algorismes intenten aconseguir malles de triangles que representin la superfície de forma tridimensional el més acuradament possible d'una forma eficient. El requisit de l'eficiència ve donada per la interactivitat, ja que com es comenta a l'annex 12.1.3 *(Gràfics interactius vs Gràfics realistes)*, els gràfics interactius necessiten actualitzar la imatge vàries vegades per segon. Per aconseguir tots dos objectius (representació acurada i velocitat), que en principi es contradiuen, el que fa la majoria d'algorismes és generar una malla de triangles potencialment

Figura 1: Esquema de representació de terrenys

diferent per cada posició de l'observador. D'aquesta manera s'intenta generar una malla de triangles que sigui la òptima per aquesta posició, on òptima significa ser el més acurada possible per allò que és aprop de l'observador i alhora suficientment ràpida per poder-se visualitzar de forma interactiva. La posició de l'observador és clau en aquest aspecte ja que no cal tenir una gran exactitud en la representació de parts de la superfície que es trobin molt lluny de l'observador.

3.2 L'algorisme ROAM

L'algorisme ROAM *(Real-time Optimally Adapting Meshes)* és un algorisme de visualització dinàmica de terrenys. Com hem explicat en el punt anterior això significa que la malla de triangles utilitzada per representar el terreny es genera tenint en compte la posició de l'observador. A més a més, la malla va evolucionant segons l'observador es mou. ROAM utilitza arbres binaris de triangles (si teniu dubtes de que són podeu consultar la secció 12.4) per representar la superfície.

3.2.1 Prioritat dels triangles

Per tal d'implementar l'algorisme cal que cada triangle tingui una prioritat associada, així podrem decidir si és prioritari partir un triangle o un altre. El càlcul de la prioritat té dues parts. La primera part és estable respecte a la posició de l'observador mentre que la segona varia segons la posició d'aquest.

La primera part és el que s'anomena *wedgie*, i es defineix de forma recursiva. Pels triangles que són fulles de l'arbre, el wedgie és el valor absolut de la diferència entre el punt que hauria d'ocupar en la realitat el punt mig de la base del triangle i el punt

que està ocupant en la representació. És a dir, el valor absolut de la resta del valor que obtenim de demanar-li al mapa l'alçada del punt mig de la base del triangle i de la mitja de les alçades dels dos vèrtex extrems de la base. A la figura 2 podem veure una representació esquemàtica en 2D de que és el wedgie, la línia negra és el terreny real, la vermella l'aresta base del triangle que el representa i la verda el wedgie d'aquest triangle.

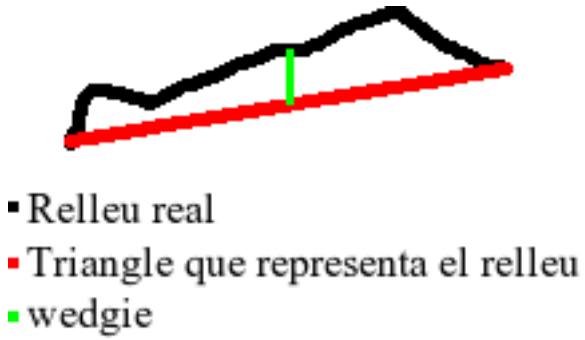

Figura 2: Esquema en 2d del wedgie

Per aquells triangles que no són fulla de l'arbre, el wedgie del triangle és la suma del càlcul anterior i el màxim dels wedgies dels dos fills del triangle.

La segona part del càlcul de la prioritat consisteix en ponderar el wedgie segons la posició de l'observador per tal de poder tenir un algorisme dinàmic. Aquesta ponderació és lògica ja que si pensem en el wedgie com a error que està cometent un triangle, no ens ha d'importar que hi hagi errors grans si aquests estan molt lluny de l'observador, mentre que potser sí hauriem d'intentar corregir un error petit si aquest està just al davant de l'observador. La ponderació que es fa servir per calcular el valor final de la prioritat d'un triangle és la projecció del wedgie a espai de pantalla, amb això fem un càlcul proporcional al nombre de píxels d'error que el wedgie realment està causant a la pantalla, i per tant, tenim un càlcul que ens pot servir per ordenar els triangles tenint en compte quin és el que està causant més error a la visualització.

3.2.2 Construcció de l'arbre de triangles

Hi ha dues formes de construir l'arbre de triangles donat un mapa i una posició de l'observador. La primera reconstrueix l'arbre per a cada nova posició de l'observador, això la fa menys eficient però més fàcil d'entendre, el segon mètode reutilitza l'arbre de la posició anterior de l'observador i el transforma en un arbre òptim per la nova posició.
La primera forma de construir l'arbre de triangles utilitza aquest algorisme.

```
Sigui Q una cua de triangles ordenada per prioritat
Afegir els dos triangles del diamant base del mapa a Q
Mentre el nombre de triangles del mapa sigui massa petit fer {
  Obtenir T, el triangle de Q amb més prioritat
  Partir T
  Actualitzar Q d'aquesta forma {
    Eliminar T i els altres triangles partits de Q
    Afegir tots els triangles creats per la partició a Q
  }
}
```

D'aquesta manera es garanteix que quan arribem al nombre de triangles al que volem que arribi l'algorisme tindrem l'arbre òptim, ja que hem anat partint sempre els triangles que tenien més prioritat, és a dir, aquells que cometien més error.

La segona manera de construir l'arbre de triangles és aquesta:

```
Sigui Q una cua de triangles ordenada per prioritat
Sigui K una cua de diamants ordenada per prioritat
Si estem al instant inicial
  Afegir els dos triangles del diamant base del mapa a Q
Altrament
  Actualitzar la prioritat de tots els elements de Q i K donada la nova
  posició de l'observador
Mentre el nombre de triangles del mapa sigui massa petit o la prioritat
màxima de Q sigui més gran que la mínima de K fer {
  Si el nombre de triangles del mapa és massa petit {
    Obtenir T, el triangle de Q amb més prioritat
    Partir T
    Actualitzar les cues d'aquesta forma {
      Eliminar T i els altres triangles partits de Q
      Afegir tots els triangles creats per la partició a Q
      Eliminar de K qualsevol diamant del que s'hagin partit els seus fills
      Afegir a K tots els nous diamants fusionables que s'hagin creat
  } Altrament {
    Obtenir D, format per T i T2, el diamant de K amb menys prioritat
    Fusionar D
    Actualitzar les cues d'aquesta forma {
      Eliminar de Q els fills de T i T2, fusionats al fusionar D
      Afegir T i T2 a Q
      Eliminar D de K
      Afegir a K tots els nous diamants fusionables que s'hagin creat
  }
}
```

Si ens fixem en el cas de l'instant inicial el codi es comporta de forma idèntica al primer algorisme, ja que el mapa sempre té massa pocs elements i per tant entra al primer cas del *Si*. A la resta de casos es reaprofiten els triangles bons de l'arbre de triangles que s'havia creat per la posició anterior de l'observador, mentre que aquells

triangles que havien estat partits anteriorment, formant diamants fusionables, i que han passat a tenir menys prioritat que triangles que es poden partir, es fusionen. Així a la propera volta del *Mentre*, hi haurà menys triangles dels necessaris, s'entrarà al primer cas del *Si* i es partirà el triangle que estigui cometent més error.

3.2.3 View frustum culling

El view frustum culling no és un requeriment obligatori de l'algorisme ROAM, però si s'implementa milloren els resultats. La implementació del view frustum culling consisteix en calcular per cada triangle si està o no dintre del volum de visualització, és a dir, dintre de tot allò que s'està mostrant per pantalla. Si un triangle no està dintre del volum de visualització se li assigna una prioritat molt petita, ja que encara que s'estigui cometent un gran error en aquell triangle, realment a l'observador no li importa, ja que no està veient aquella zona.

3.3 El format DEM

El format DEM (Digital Elevation Model) va ser creat per l'USGS *(United States Geological Survey)* per guardar la informació de les seves mesures de la superfície dels Estats Units.

Un fitxer DEM està organitzat en tres tipus de registres lògics: A, B i C.

- El tipus de registre A conté informació en quant a les característiques generals del mapa, nom, límits, unitats de mesura, elevació mínima i màxima, el número de registres de tipus B, etc. Només hi ha un registre de tipus A per fitxer.
- Els registres de tipus B contenen la informació de les alçades.
- El tipus de registre C conté estàdistiques en quant a l'exactitud de les dades del fitxer.

El format DEM estava pensat per emmagatzemar-se en cintes magnètiques i és per això que l'especificació del format parla de registres físics, cadascun de 1024 bytes. En un registre físic no hi pot haver més d'un registre lògic, en canvi és usual que els registres lògics de tipus B ocupin més d'un registre físic. Les parts d'un registre físic no ocupades per la informació d'un registre lògic s'han d'omplir amb blancs. Els registres lògics de tipus A i C mai ocupen més d'un registre físic.

Els fitxers DEM divideixen el mapa en una graella i proporcionen l'alçada de cadascun dels punts de la graella. Cada registre de tipus B té la informació de les alçades d'una columna d'aquesta graella, és a dir, si la graella d'un mapa tingués n columnes i m files hi hauria n registres de tipus B cadascun amb m alçades.

Hi ha tres tipus de fitxers DEM:

1. **DEM d'1 grau:** Tenen 1201*1201 punts de dades, estan basats en latitud-longitud i l'espaiat entre els punts de la graella és de 3 segons d'arc, que és aproximadament 90 metres, encara que obviament depèn de la latitud del planeta en que s'hagin pres les mesures.

2. **DEM de 30 minuts:** Són 4 unitats de fitxers DEM d'1 grau distribuïdes de forma conjunta, són els menys comuns dels 3 tipus.

3. **DEM de 7.5 minuts:** L'espaiat entre els punts de la graella pot ser de 30 o 10 metres (1 segon d'arc i 1/3 de segon d'arc respectivament), els límits del fitxer estan basats en latitud-longitud, però en canvi els punts de dades es donen en coordenada x-y d'una projecció Mercator, resultant en fitxers on no tots els registres de tipus B tenen la mateixa longitud. (Figura 3)

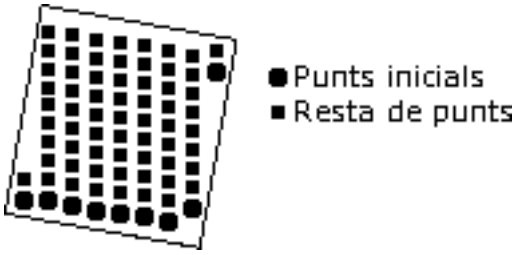

Figura 3: Esquema d'un DEM de 7.5 minuts

3.3.1 Per què el format DEM?

Hi ha molts tipus de format de fitxer diferents que descriuen terrenys. Aquesta secció explica quines són les raons bàsiques que han portat a escollir aquest tipus de format i no un altre.

- **Disponibilitat de l'especificació:** Per tal de poder implementar un parser d'un tipus de format de fitxer és molt important que hi hagi disponibilitat de l'especificació, ja que sense especificació s'ha de fer enginyeria inversa sobre el fitxer i això, a part d'estar fora del que es vol fer amb el projecte, hauria allargat de forma considerable el temps de desenvolupament. Per implementar el parser de DEM s'ha fer servir el document sobre DEM que es menciona a la bibliografia.

- **Disponibilitat de mapes:** La disponibilitat de mapes és, evidentment, molt important. Sense mapes no val per a res implementar el parser ja que no es pot comprovar que la implementació sigui correcte, a més a més, no es pot verificar tampoc el funcionament de l'algorisme ROAM. A la bibliografia es pot trobar una pàgina on hi ha mapes de Hawaii en format DEM.

3.4 El format de mapes simples

El format de mapes simples *(extensió PM)* és un format inventat per mi. El vaig fer servir per poder provar les primeres implementacions de l'algorisme abans d'haver desenvolupat el lector de fitxers DEM. Els mapes simples tenen una sintaxi molt simple, la primera línia conté un nombre k que representa el nombre de files i columnes del mapa (els mapes han de ser quadrats), a partir d'aquí hi ha $k * k$ nombres, que representen les k files del mapa, on cada fila té k elements.

4 Especificació

4.1 Què és libroam?

libroam és la part del projecte que implementa l'algorisme ROAM. Està programada com a llibreria de forma que qualsevol programa pugui usar aquesta funcionalitat. La llibreria està sota llicència GPL (l'annex 12.5 explica que és aquesta llicència). La llibreria funciona fent servir el paradigma de navegació fly. Per veure una descripció dels diferents paradigmes de navegació aplicables als gràfics 3D consulteu l'annex 12.3

4.2 Què és r3v?

r3v és una aplicació que utilitza la llibreria libroam per tal de proporcionar un visor de superfícies que utilitzi l'algorisme ROAM. L'aplicació afegeix sobre la llibreria una interfície gràfica d'usuari basada en la llibreria Qt (A l'annex 12.6 trobareu una descripció d'aquesta llibreria) que permet a l'usuari obrir i tancar mapes, així com visualitzar-los i navegar per ells.

4.3 Requeriments no funcionals

Definició: Els requeriments no funcionals (o qualitats del sistema), capturen les propietats requerides pel sistema software, com el rendiment, la seguretat, la mantenibilitat, etc., en altres paraules, com de bé s'ha de complir un aspecte estructural o de comportament del sistema.

- **Traduibilitat:** Que la interfície gràfica del programa es pugui traduir a tots els idiomes que es desitgi és un requeriment important, així s'amplia el públic potencial al qual anirà dirigida l'aplicació, ja que no tothom té coneixement d'anglès.

- **Portabilitat:** Es vol que tant r3v com libroam es puguin compilar, i per tant, executar a la majoria de sistemes disponibles.

- **Independència entre la interfície i l'algorisme:** Es vol que la implementació de la interfície gràfica sigui el més independent possible de l'algorisme ROAM, per tal que si en algun moment es vol fer una interfície gràfica usant un altre toolkit que no sigui Qt es pugui reutilitzar la majoria del codi.

- **Usabilitat:** La usabilitat és molt important en tots els projectes de software, ja que a part que un programa sigui potent en el que fa, és necessari que sigui fàcil d'usar.
- **Mantenibilitat:** És necessari que el programa es pugui mantenir amb relativa facilitat, ja que normalment el cicle de vida dels projectes de software porta als programadors a revisar/ampliar el codi.
- **Programari lliure:** Es vol que l'aplicació sigui programari lliure, en especial, programari lliure sota la llicència GPL.

4.4 Requeriments funcionals

Definició: Els requeriments funcionals capturen el comportament que es vol que tingui el sistema software. Aquest comportament es pot expressar com a serveis, tasques o funcions que cal que el sistema sigui capaç de fer.

- **Implementar l'algorisme ROAM:** Volem que el projecte implementi l'algorisme ROAM per tal de visualitzar terrenys.
- **Navegar pels terrenys:** Es vol que el programa permeti a l'usuari navegar pel terreny. S'ha escollit el paradigma de navegació fly.
- **Carregar mapes des de fitxer:** No es vol que el terreny que visualitzi el programa sigui fixe, per tant, s'haurà d'implementar la possibilitat d'obrir nous mapes des de fitxers.
- **Intèrpret de fitxers DEM:** El requeriment anterior implica que caldrà saber llegir algun tipus de fitxer que proporcioni les dades de les alçades per crear el mapa, com s'explica a l'apartat 3.3 s'ha decidit utilitzar el format DEM.

4.5 Model Conceptual

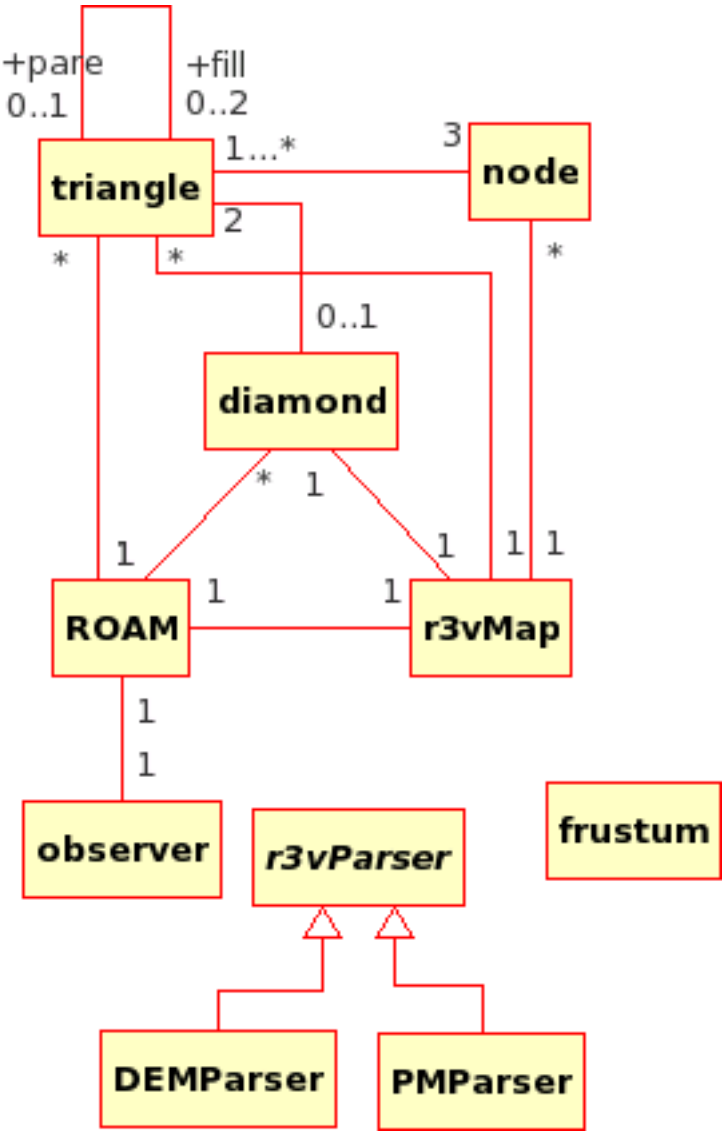

Figura 4: Diagrama de classes del domini

4.5.1 Restriccions d'integritat textuals

- Un triangle o bé té dos fills o no en té cap, no pot tenir-ne només un.

4.5.2 Classe r3vMap

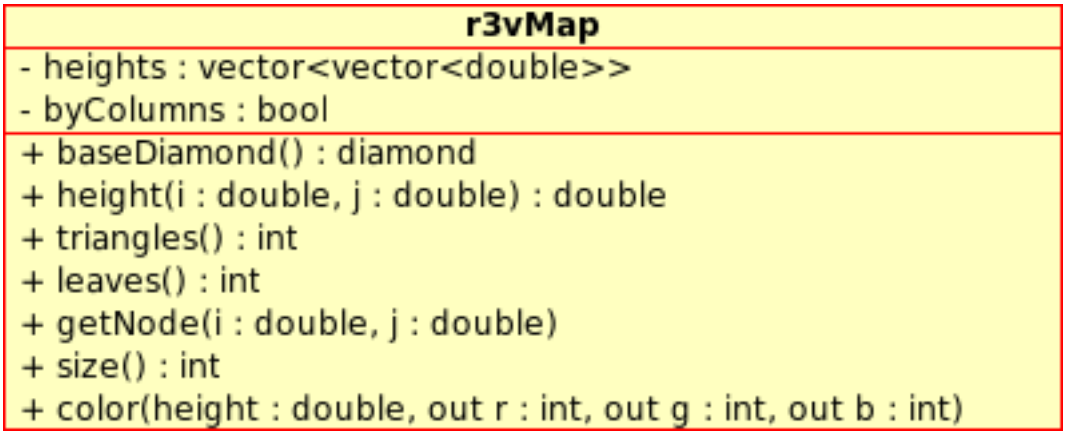

Figura 5: Classe r3vMap

Descripció informal: La classe *r3vMap* és la classe encarregada d'emmagatzemar la informació dels mapes, per aquest fi té els següents membres i operacions:

- *heights*: emmagatzema les alçades de tots els punts disponibles al mapa en forma de vector de vectors.
- *byColumns*: indica si aquest vector de vectors és un vector de files o un vector de columnes.
- *baseDiamond*: retorna el diamant que forma la base del mapa
- *height*: retorna l'alçada del punt donat
- *triangles*: retorna el nombre de triangles que hi ha representant el mapa actualment
- *leaves*: retorna el nombre de fulles que hi ha representant el mapa actualment
- *getNode*: retorna el node que representa el punt donat
- *size*: retorna la mida dels costats del mapa, és a dir, la longitud de cadascun dels vectors de *heights* que és la mateixa que la del vector *heights*, ja que els mapes són quadrats
- *color*: retorna el color que ha de representar l'alçada donada

4.5.3 Classe diamond

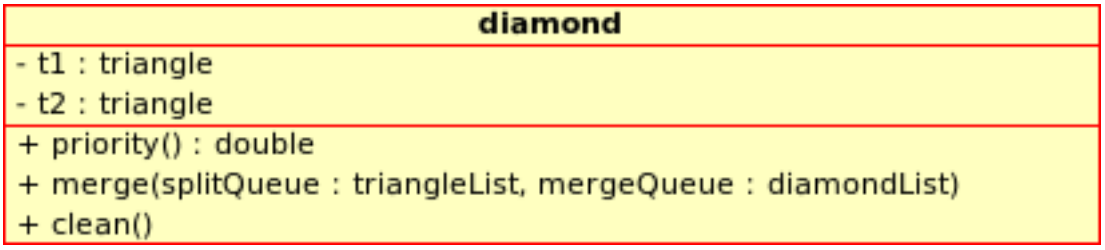

Figura 6: Classe diamond

Descripció informal: La classe *diamond* representa la unió de dos triangles que comparteixen la seva base en un diamant. S'utilitza per dues funcions: obtenir el diamant base del mapa per tenir així accés al bosc de triangles que formen el mapa i per mantenir la llista de diamants potencialment fusionables que necessita l'algorisme. Descripció dels membres:

- $t1$: un dels triangles del diamant.
- $t2$: l'altre triangle del diamant.
- *priority*: retorna la prioritat del diamant, que és la màxima de la prioritat dels dos triangles que el formen.
- *merge*: fusiona el diamant actualitzant les llistes de triangles partibles i diamants fusionables.
- *clean*: elimina els triangles que formen el diamant, només s'utilitza quan es tanca el mapa.

Hi ha diamants que només estan formats per un triangle, es tracta de triangles la base dels quals pertany al marge del mapa, és a dir, no tenen possibilitat de tenir triangle base. És necessari que aquests triangles també crein el seu propi diamant perquè sinó l'algorisme no els tindria en compte per fusionar-los i un cop partits sempre s'hi quedarien.

4.5.4 Classe frustum

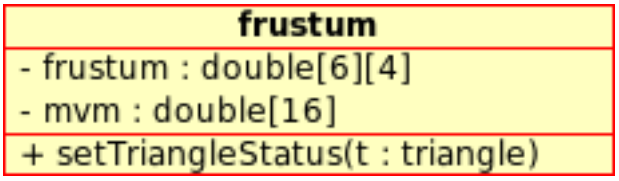

Figura 7: Classe frustum

Descripció informal: La classe *frustum* representa el frustum de visualització actual, també emmagatzema la matriu modelViewMatrix, que és necessària per calcular el frustum i també per calcular la prioritat dels triangles. Descripció dels membres:

- *frustum*: conté els quatre coeficients dels sis plans que formen el frustum de visualització actual.
- *mvm*: conté els setze valors de la matriu modelViewMatrix.
- *setTriangleStatus*: estableix l'estat del triangle respecte al frustum. L'estat pot ser: completament dintre, completament fora o parcialment dintre del frustum.

4.5.5 Classe observer

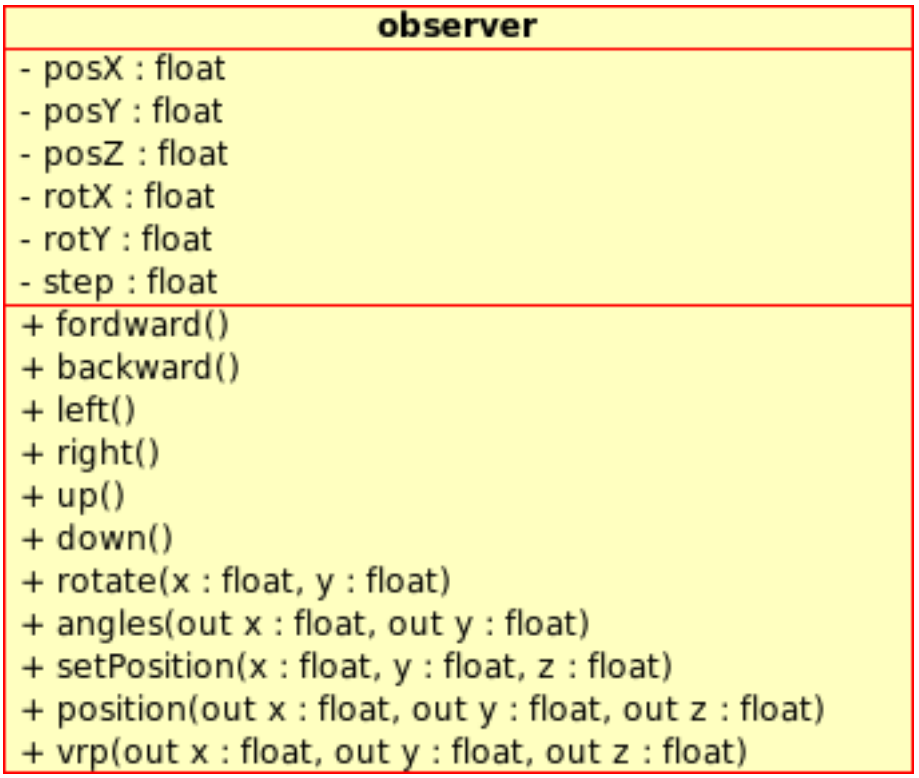

Figura 8: Classe frustum

Descripció informal: La classe *observer* representa l'observador. Membres:

- $posX, posY, posZ$: la posició del observador.
- $rotX, rotY$: la rotació de l'observador respecte els eixos, defineix la direcció de visió.
- *step*: longitud del pas de l'observador.
- $fordward$: mou a l'observador un pas endavant en la direcció de visió.
- *backward*: mou a l'observador un pas enrera en la direcció de visió.
- $left$: mou a l'observador un pas a l'esquerra respecte la direcció de visió.
- *right*: mou a l'observador un pas a la dreta respecte la direcció de visió.
- *up*: mou a l'observador un pas amunt respecte el mapa.
- *down*: mou a l'observador un pas avall respecte el mapa.

- *rotate*: canvia la rotació de l'observador.
- *angles*: retorna els angles de rotació actuals.
- *position*: retorna la posició actual.
- *setPosition*: estableix la posició de l'observador.
- *vrp*: retorna el vrp (View Reference Point) de l'observador.

4.5.6 Classe node

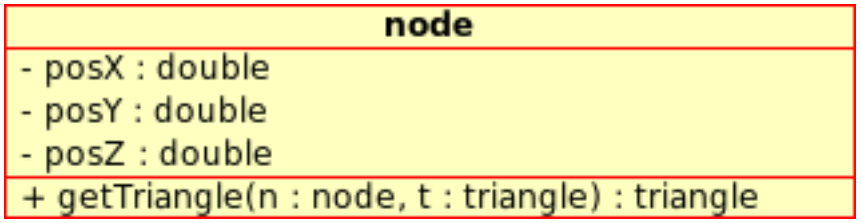

Figura 9: Classe node

Descripció informal: La classe *node* representa un vèrtex d'un triangle. Descripció dels membres:

- $posX, posY, posZ$: posició del vèrtex
- $getTriangle$: retorna el triangle que comparteixi aquest node i el node donat amb el triangle donat. En cas que n'hi hagi més d'un (n'hi pot haver dos), retorna el de nivell més petit. Aquesta operació es fa servir per obtenir el triangle que comparteix la base d'un triangle; per fer-ho es crida l'operació sobre el node esquerre d'un triangle i es passen com a paràmetre el node dret del triangle i el triangle.

4.5.7 Classe triangle

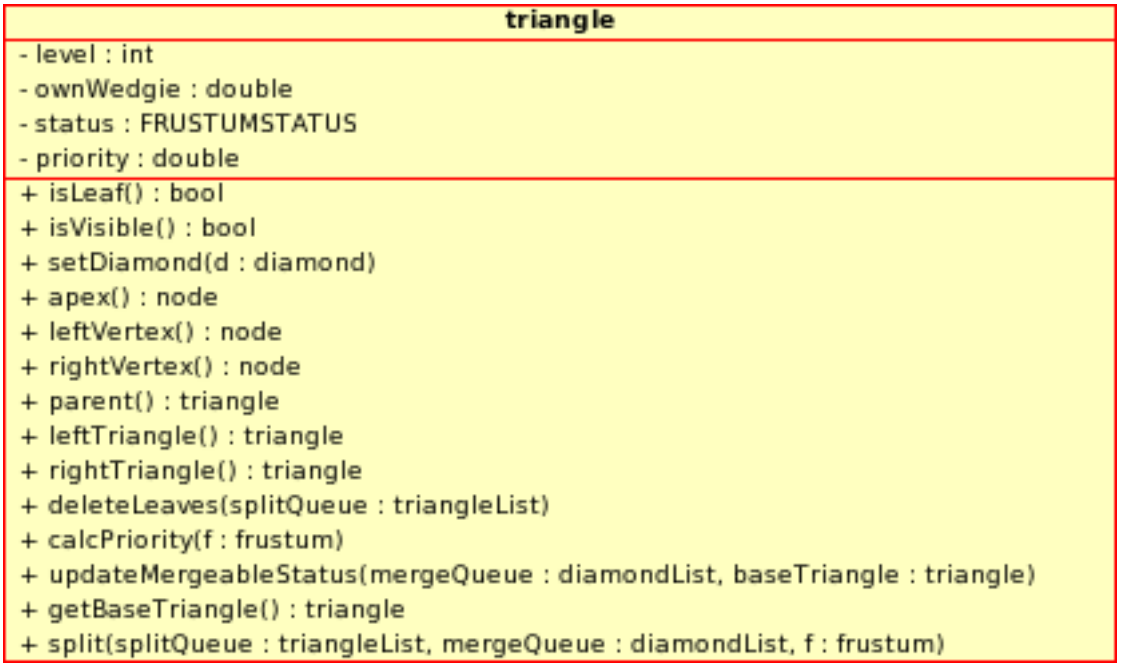

Figura 10: Classe triangle

Descripció informal: La classe *triangle* representa un triangle dintre de la malla de triangles formada per la visualització del mapa. Descripció dels membres:

- *level*: el nivell del triangle. És el nombre de triangles pare que s'han de recórrer per arribar al diamant base.
- $ownWedgie$: el wedgie d'aquest triangle.
- *status*: defineix l'estat del triangle respecte el frustum, pot ser: completament fora, completament dintre o parcialment dintre del frustum de visualització.
- *priority*: la prioritat d'aquest triangle.
- $isLeaf$: retorna cert si el triangle no té fills.
- $isVisible$: retorna cert si el triangle és visible, és a dir, esta completa o parcialment dintre del frustum.
- $apex, leftVertex, rightVertex$: retornen l'àpex, el vèrtex esquerre i el vèrtex dret del triangle respectivament.

- *parent*: retorna el triangle pare (en cas que n'hi hagi).
- $leftTriangle, rightTriangle$: retornen el fill esquerre i dret respectivament (en cas que existeixin).
- *deleteLeaves*: elimina els triangles fills en cas que n'hi hagi actualitzant la llista de triangles partibles.
- *calcPriority*: actualitza el camp *priority* tenint en compte la posició actual de l'observador (matriu de modelViewMatrix proporcionada pel paràmetre $frustum$).
- $updateMergeable$: actualitza la llista de diamants fusionables. De forma que elimina els diamants que ja no són fusionables perquè s'han partit els seus fills i afegeix els nous diamants fusionables quan els fills dels triangles no tenen més fills.
- $getBaseTriangle$: retorna el triangle que comparteix la base amb el triangle.
- *split*: parteix el triangle actualitzant les llistes de triangles partibles i diamants fusionables.

4.5.8 Classe ROAM

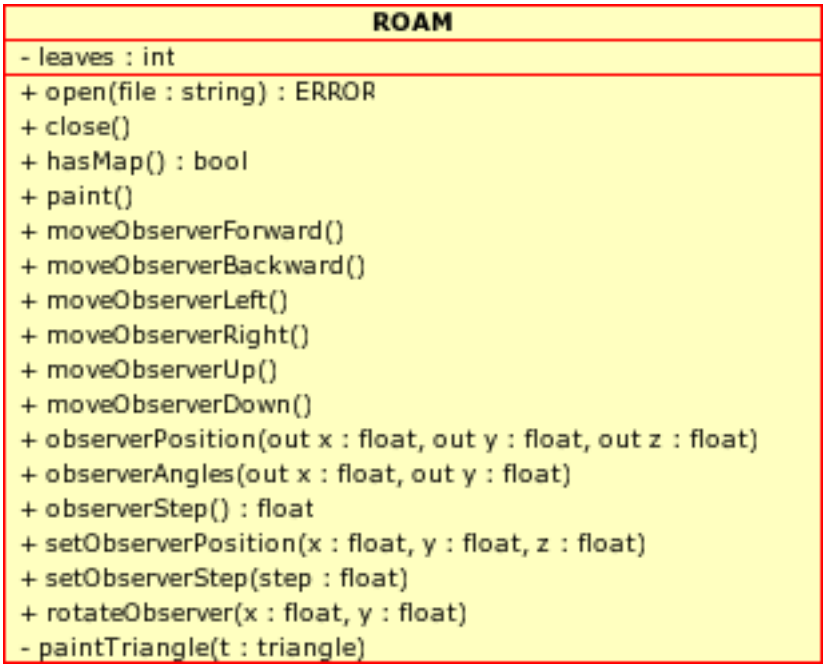

Figura 11: Classe ROAM

Descripció informal: La classe *ROAM* és on es troba la implementació de l'algorisme ROAM. Descripció dels membres:

- *leaves*: nombre de fulles que volem que tingui l'execució de l'algorisme sobre el mapa actual.
- *open*: obre el mapa que es troba al fitxer donat, en cas que hi hagi algun error retorna un codi d'error explicant la possible causa.
- *close*: tanca el mapa actual.
- *hasMap*: retorna si hi ha algun mapa obert actualment.
- *paint*: és on es centra tot l'algorisme ROAM. Actualitza les llistes de triangles partibles i diamants fusionables donada la nova posició de l'observador i parteix triangles i fusiona diamants fins obtenir la malla de triangles òptima per aquesta posició.
- *moveObserverForward*: mou l'observador cap endavant
- *moveObserverBackward*: mou l'observador cap enrera

- *moveObserverLeft*: mou l'observador cap a l'esquerra
- *moveObserverRight*: mou l'observador cap a la dreta
- *moveObserverUp*: mou l'observador cap a dalt
- *moveObserverDown*: mou l'observador cap a baix
- *observerPosition*: retorna la posició de l'observador
- *observerAngles*: retorna els angles que determinen la línia de visió de l'observador
- *setObserverPosition*: estableix la posició de l'observador
- *rotateObserver*: rota la línia de visió de l'observador
- *paintTriangle*: pinta el triangle donat en cas que sigui un triangle fulla, si no fa una crida recursiva per pintar els seus dos fills.

4.5.9 Classes parser

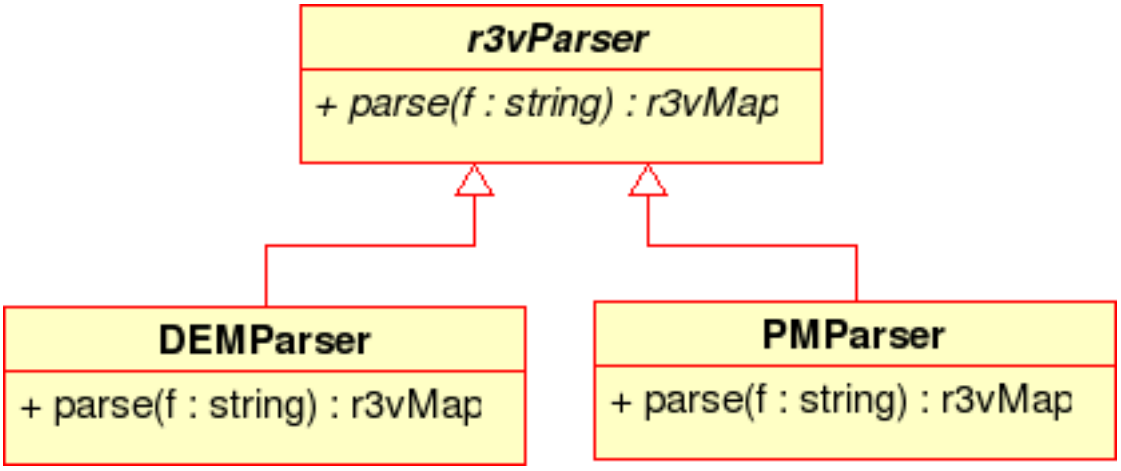

Figura 12: Classes parser

Aquestes classes són les encarregades d'interpretar fitxers que representen mapes i retornar la representació en forma de $r3vMap$.

- ***r3vParser*** és una classe abstracta que defineix l'operació *parse* que tots els intèrprets de mapes han d'implementar.
- **DEMParser** és una classe que implementa un intèrpret de mapes en format DEM.
- **PMParser** és una classe que implementa un intèrpret de mapes en format PM (mapa simple).

4.6 Model de casos d'ús

4.6.1 Diagrama de casos d'ús

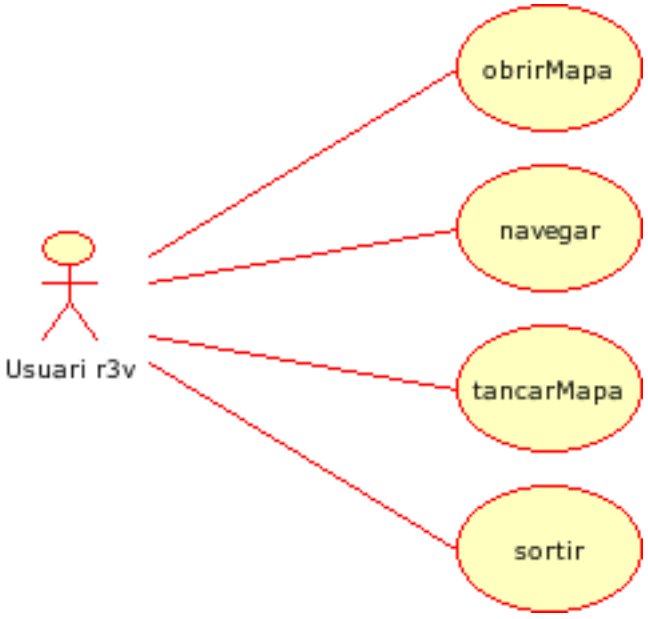

Figura 13: Diagrama de casos d'ús

4.6.2 Cas d'ús obrirMapa

Cas d'ús: obrirMapa
Actors: Usuari r3v
Propòsit: Obrir un mapa per tal de visualitzar-lo
Tipus: Primari essencial
Curs típic d'esdeveniments:

Accions dels actors	**Resposta del sistema**
1. L'usuari prem l'opció d'obrir un mapa	
	2. El sistema mostra els fitxers que es poden obrir
3. L'usuari selecciona quin és el fitxer que vol obrir	
	4. El sistema obre el mapa contingut en aquell fitxer i el mostra

4.6.3 Cas d'ús navegar

Cas d'ús: navegar
Actors: Usuari r3v
Propòsit: Navegar per la visualització del mapa
Tipus: Primari essencial
Curs típic d'esdeveniments:

Accions dels actors	Resposta del sistema
1. L'usuari prem alguna de les tecles que permet navegar per la visualització	
	2. El sistema calcula la nova posició de l'observador, actualitza la malla de triangles i actualitza la visualització mostrada per pantalla

4.6.4 Cas d'ús tancarMapa

Cas d'ús: tancarMapa
Actors: Usuari r3v
Propòsit: Tancar el mapa que s'està visualitzant en aquest moment
Tipus: Primari essencial
Curs típic d'esdeveniments:

Accions dels actors	Resposta del sistema
1. L'usuari prem l'opció de tancar el mapa	
	2. El sistema tanca el mapa i actualitza la visualització de forma que no mostri res

4.6.5 Cas d'ús sortir

Cas d'ús: sortir
Actors: Usuari r3v
Propòsit: Sortir de l'aplicació
Tipus: Primari essencial
Curs típic d'esdeveniments:

Accions dels actors	Resposta del sistema
1. L'usuari prem l'opció de sortir	
	2. El sistema es tanca a si mateix

5 Disseny

5.1 Arquitectura en 3 capes

En un sistema software és necessari descomposar les tasques del sistema en vàries capes per tal que cada capa s'ocupi de les tasques que estiguin en un nivell determinat d'abstracció. La separació típica és en tres capes, una que s'ocupi de la visualització (Capa de presentació), una que s'ocupi d'implementar la funcionalitat del sistema (Capa del domini) i una altra que s'ocupi d'interaccionar amb els fitxers (Capa de gestió de dades).

La capa de gestió de dades està formada per les classes $r3vParser$, $DEMParser$ i $PMParser$. Aquestes classes són les encarregades d'obrir i interpretar els fitxers que representen mapes de forma que la capa del domini sigui capaç d'entendre les dades que contenen.

La capa de domini de dades està formada per tota la resta de classes que apareixen al diagrama de classes del domini (Figura 4).

La capa de representació té una única classe anomenada $glWidget$.

5.2 Disseny de la capa de presentació

La classe $glWidget$ és la única classe que forma la capa de presentació. Aquesta classe s'encarrega de proporcionar una vista on es pugui pintar el resultat de processar el mapa usant l'algorisme ROAM així com de capturar els esdeveniments de teclat i de ratolí i passar-los a la capa de domini per tal que aquesta faci el processat oportú.

El mode de representació utilitzat és una finestra a pantalla completa ja que d'aquesta forma s'obté una superfície de representació del mapa el més gran possible i, a més a més, l'experiència de l'usuari és més immersiva.

La interacció de l'usuari amb l'aplicació es fa mitjançant el teclat i el ratolí. El ratolí serveix per canviar la direcció de visió de l'observador i el teclat serveix per moure l'observador.

Els controls són:

- *Fletxa amunt o W*: Mou l'observador endavant.
- *Fletxa avall o S*: Mou l'observador endarrera.
- *Fletxa esquerra o A*: Mou l'observador a l'esquerre.
- *dreta o D*: Mou l'observador a la dreta.
- *Q*: Mou l'observador amunt.
- *E*: Mou l'observador avall.
- *Eix X del ratolí*: Canvia horitzontalment la direcció de visió de l'observador.
- *Eix Y del ratolí*: Canvia verticalment la direcció de visió de l'observador.

5.3 Controlador del sistema

Tot sistema software rep esdeveniments externs per part de l'usuari o per part d'altres sistemes software. Un cop s'intercepten els esdeveniments a la capa de presentació algun objecte del domini ha de ser l'encarregat de processar-los. Aquests objectes s'anomenen controladors.

5.3.1 Controlador façana

En aquest projecte s'ha optat per fer servir un controlador del tipus façana, centralitzat a la classe ROAM. L'avantatge d'utilitzar aquest tipus de controlador és que el domini és una caixa negra per la capa de presentació, d'aquesta manera, qualsevol canvi que es faci a la capa del domini, exceptuant els de la classe ROAM, no representará cap canvi per la capa de presentació, ja que tota interacció entre la capa de presentació i la capa del domini passa a través del controlador. El problema típic d'aquest tipus de controlador és que acostuma a crear classes molt grans, però degut al petit nombre d'esdeveniments que tracta el sistema aquest problema no és important en aquest projecte.

5.4 Model Conceptual

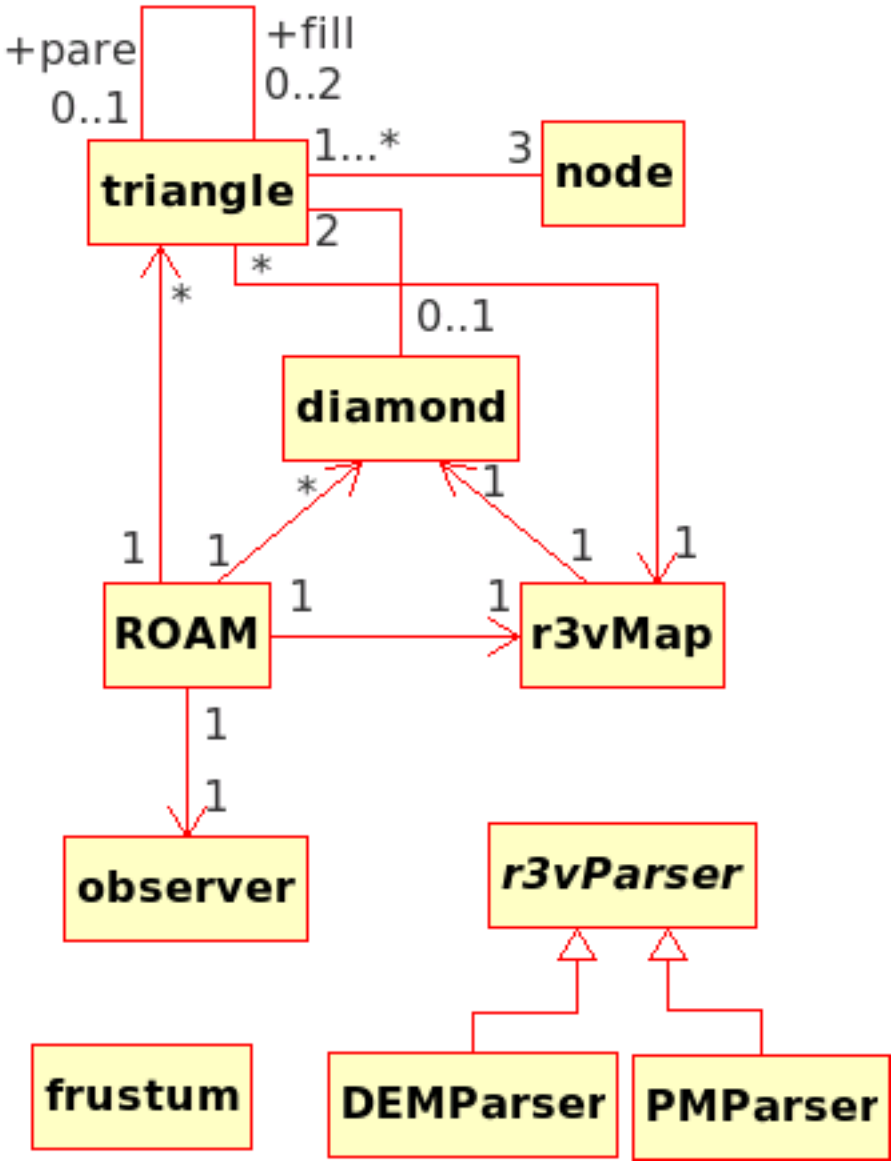

Figura 14: Diagrama de classes del domini - Disseny

Les úniques diferències que es poden observar entre aquest diagrama i el que podem veure a la Figura 4 són el canvi d'algunes associacions en associacions direccionals i la pèrdua de l'associació entre $r3vMap$ i *node* que s'ha convertit en un diccionari. Això és degut a que s'ha optat per que ningú excepte $r3vMap$ pugui crear nodes, d'aquesta manera quan un triangle necessita un node per formar-se, crida l'operació $getNode(..)$ sobre $r3vMap$, aquesta operació comprova si ja hi ha aquell node al diccionari de nodes, si el troba el retorna, si no existeix el crea, l'afegeix al

diccionari i el retorna. D'aquesta manera és molt més fàcil mantindre quins són els triangles que utilitzen un node donat.

S'han creat dues classes auxiliars, *triangleList* i *diamondList*, que emmagatzemen les relacions entre ROAM i els triangles i els diamants. Aquestes classes no estan al diagrama de classes que es troba a la part superior ja que no són més que la representació en codi de les associacions entre ROAM i triangles i diamants.

Hi ha d'altres canvis que no es veuen en aquest diagrama ja que afecten els membres de les classes. Node ha guanyat l'atribut *color* que guarda el color del node, ja que sinó caldria preguntar-li al mapa quin color havia de tenir cada cop que es volgués pintar un node i això seria molt ineficient, per tant al crear-se el node també s'emmagatzema el seu color. Per aquesta mateixa raó l'operació $color(...)$ del mapa ha passat a ser privada, ja que només es crida quan es vol crear un node.

$r3vMap$ ha guanyat els membres $minHeight$ i $amplitude$ que emmagatzemen l'alçada mínima del mapa i la diferència entre l'alçada mínima i màxima del mapa respectivament. També ha aparegut l'operació *calcAmplitude* que calcula els valors de $minHeight$ i $amplitude$. Això és per eficiència, ja que aquests valors es poden obtenir a partir del vector d'alçades, però com el seu càlcul és costós i s'usa sovint, s'ha decidit materialitzar aquesta informació derivada.

També s'han afegit a algunes classes com *triangle* i *diamant* un membre del tipus iterador, amb els seus respectius set/get, aquests camps contenen l'iterador de l'objecte dins de les llistes que manté la classe $ROAM$, això és útil ja que d'aquesta manera l'eliminació és molt més ràpida, doncs no cal tornar a buscar a la llista l'element abans d'eliminar-lo.

La classe *triangle* ha guanyat el membre *wedgie* que és el wedgie real del triangle, això és per eficiència, ja que cada vegada que necessitem el wedgie podríem usar $ownWedgie$ i fer el recorregut pels fills. També hi ha la nova operació $updateWedgie$ que serveix per notificar a un triangle que ha d'actualitzar la seva variable *wedgie*. *triangle* també ha guanyat l'operació privada $setMergeable()$ que rep els paràmetres $bool\ mergeable$, $diamondList\ *mergeQueue$, $triangle\ *baseTriangle$ i s'utilitza a $updateMergeableStatus$ per tal de no haver de repetir les tres o quatre mateixes línies de codi a tres llocs diferents de la funció.

5.5 Diagrames de seqüència

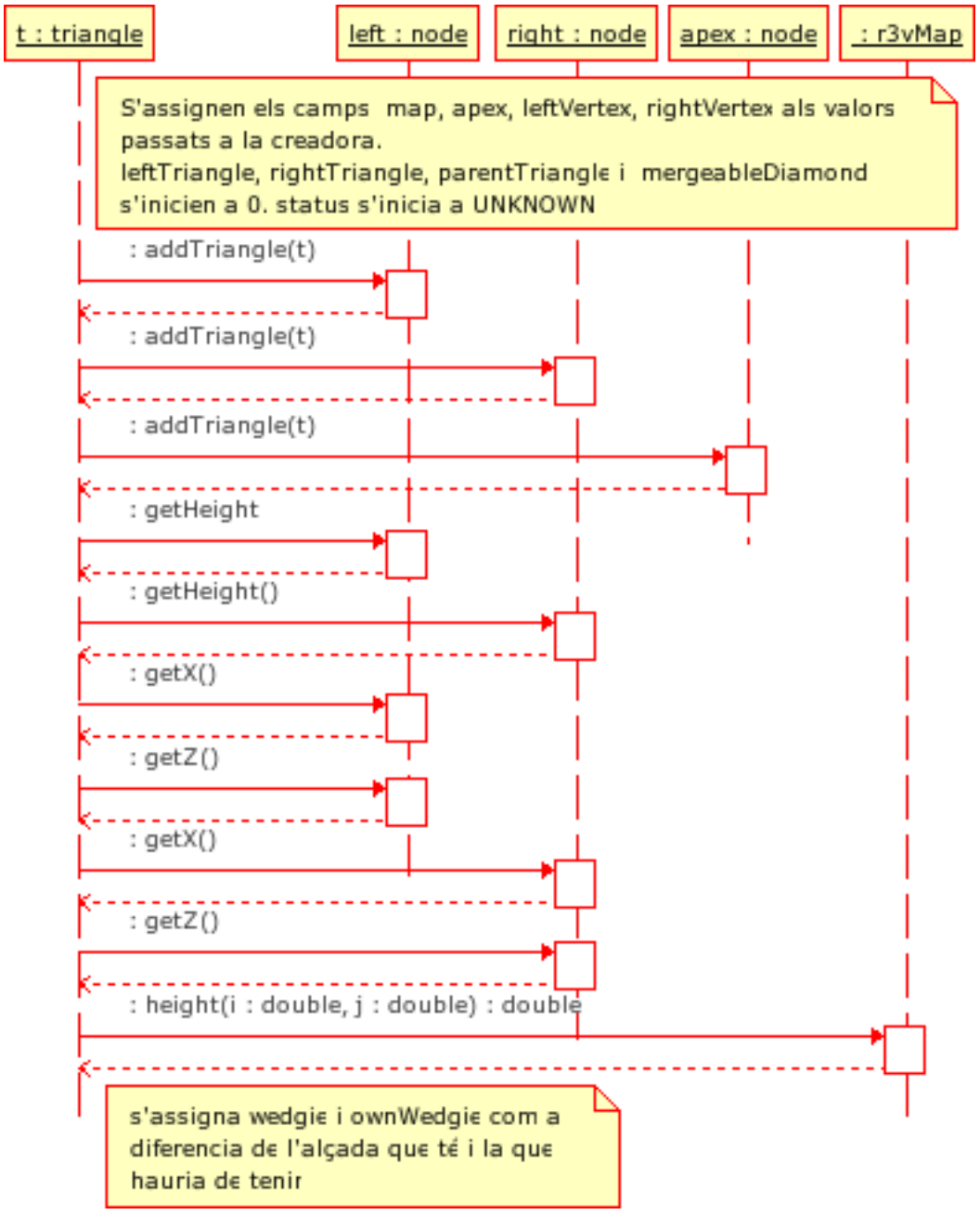

Figura 15: Diagrama de seqüència - Triangle - Creadora

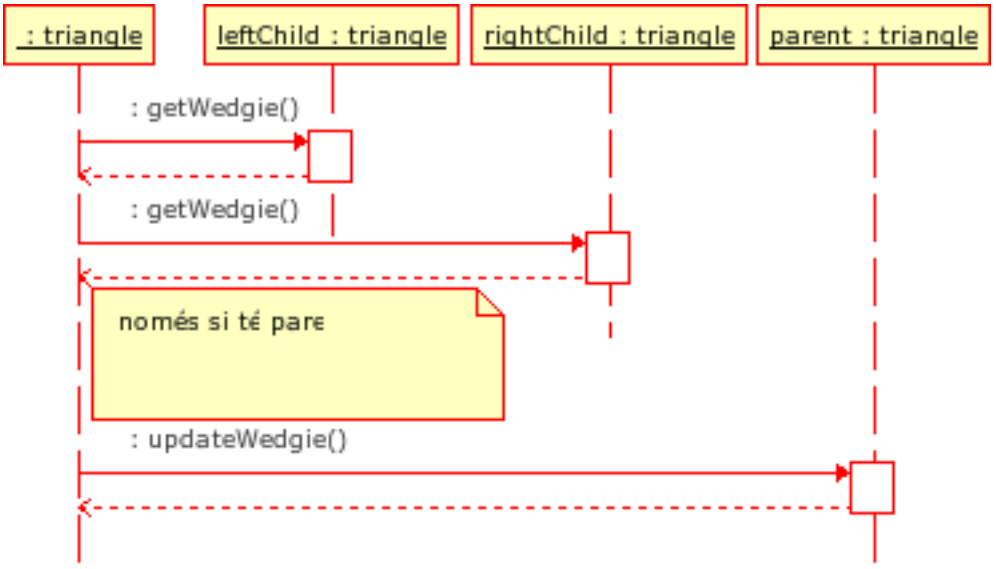

Figura 16: Diagrama de seqüència - Triangle - updateWedgie

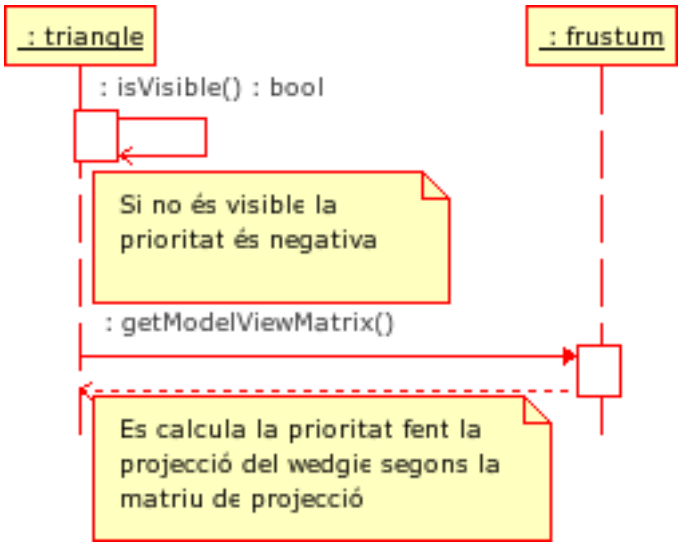

Figura 17: Diagrama de seqüència - Triangle - calcPriority

Figura 18: Diagrama de seqüència - Diamond - merge

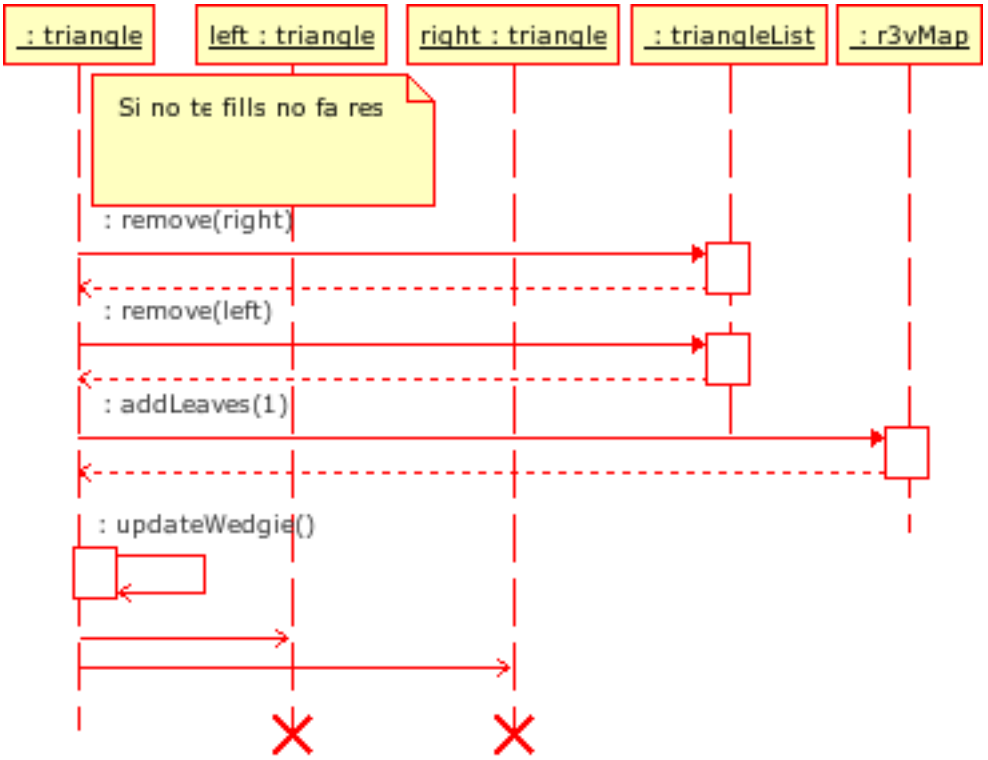

Figura 19: Diagrama de seqüència - Triangle - deleteLeaves

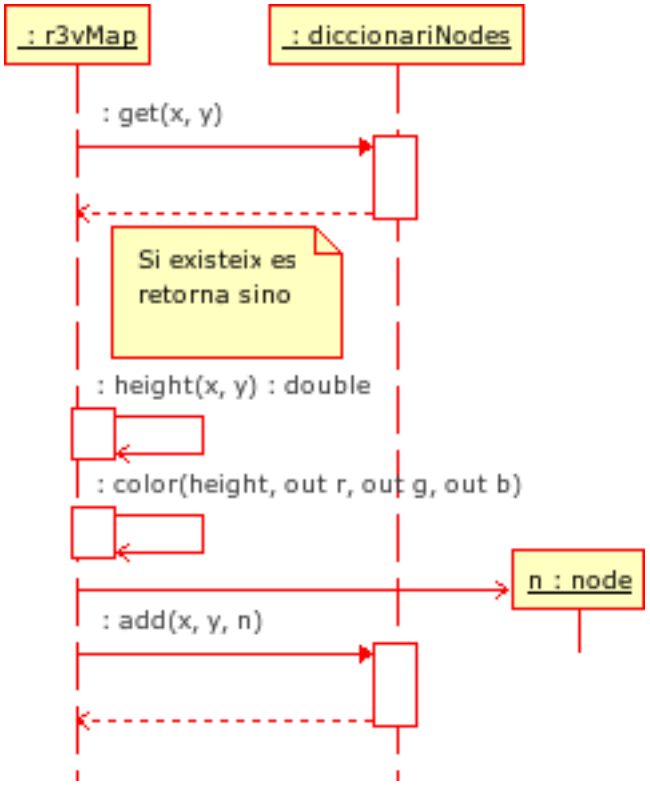

Figura 20: Diagrama de seqüència - r3vMap - getNode

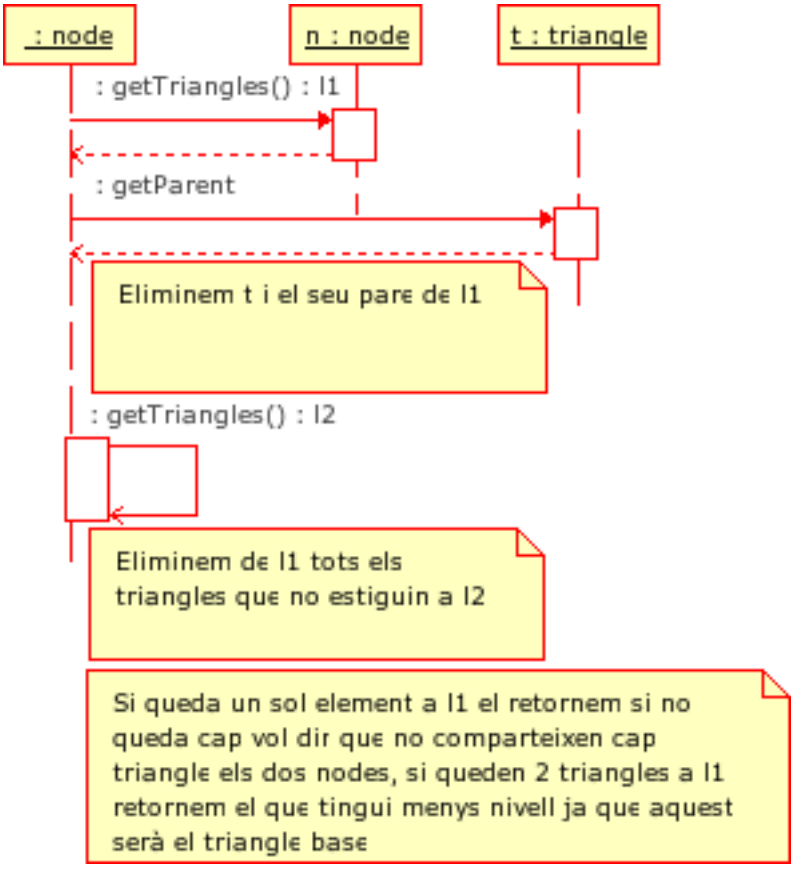

Figura 21: Diagrama de seqüència - node - getTriangle

Figura 22: Diagrama de seqüència - triangle - split

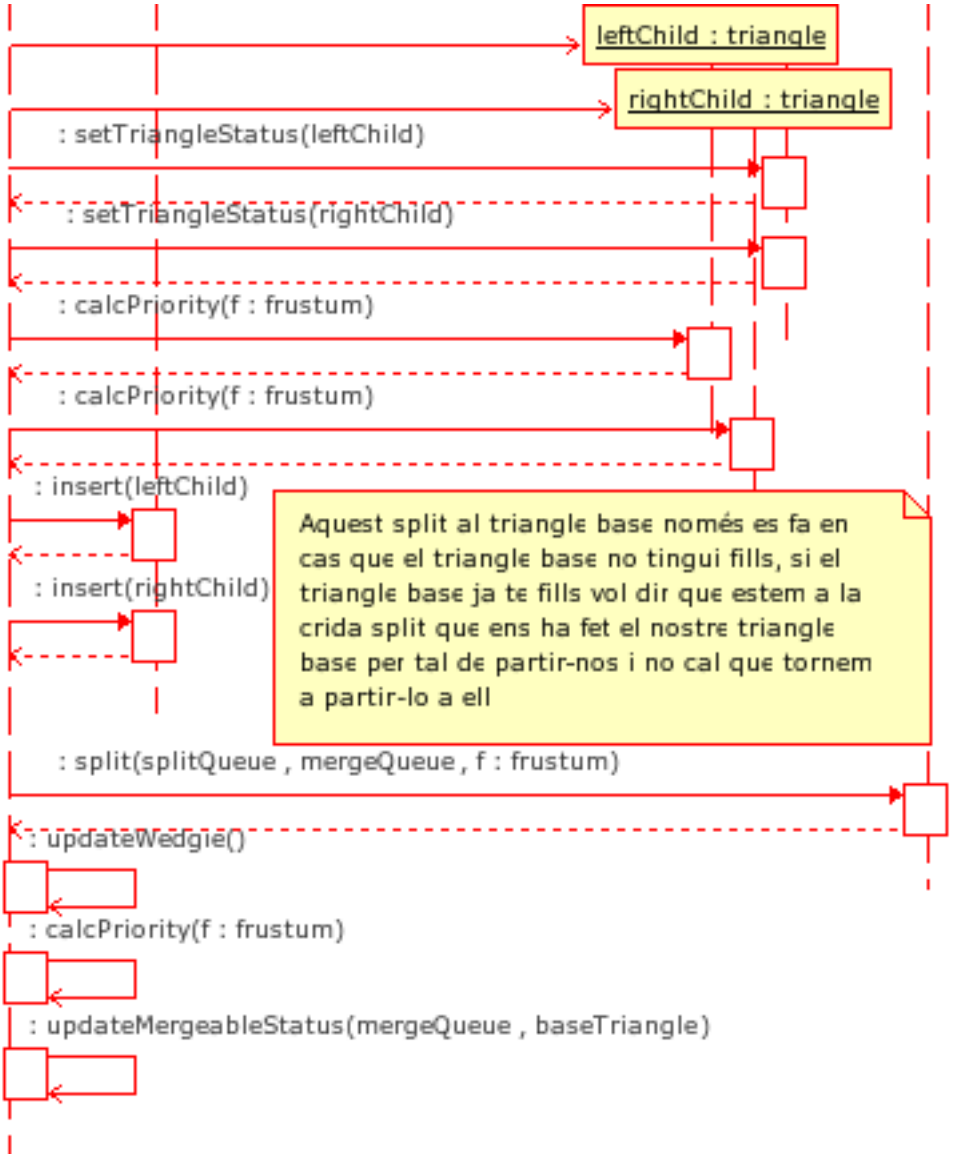

Figura 23: Diagrama de seqüència - triangle - split (continuació)

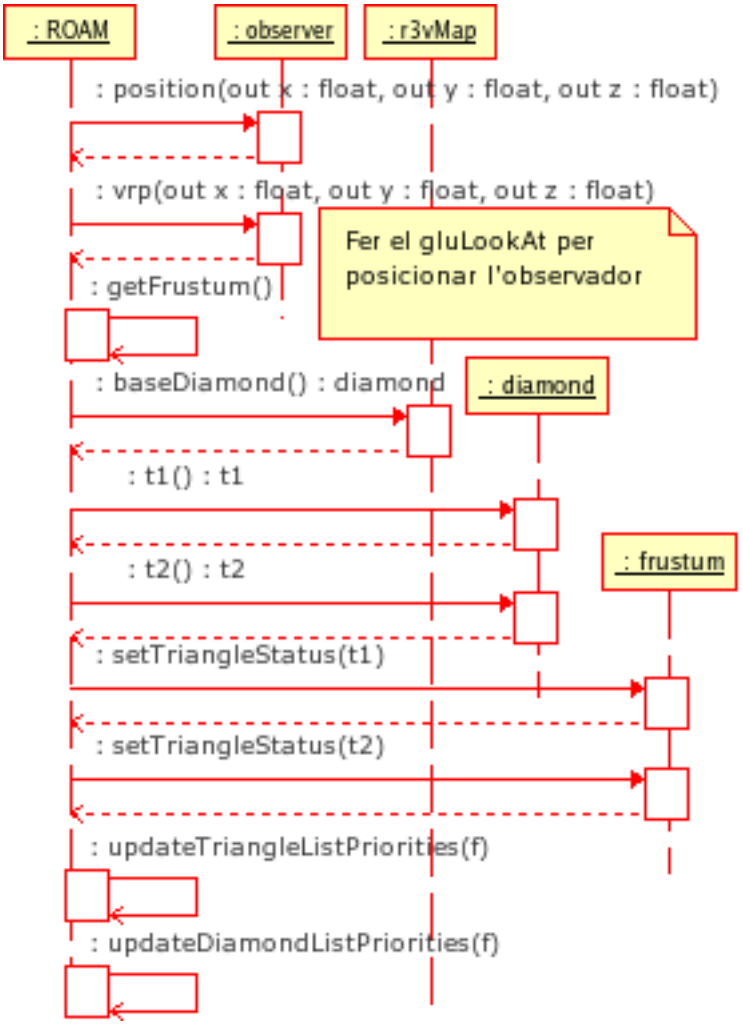

Figura 24: Diagrama de seqüència - ROAM - paint

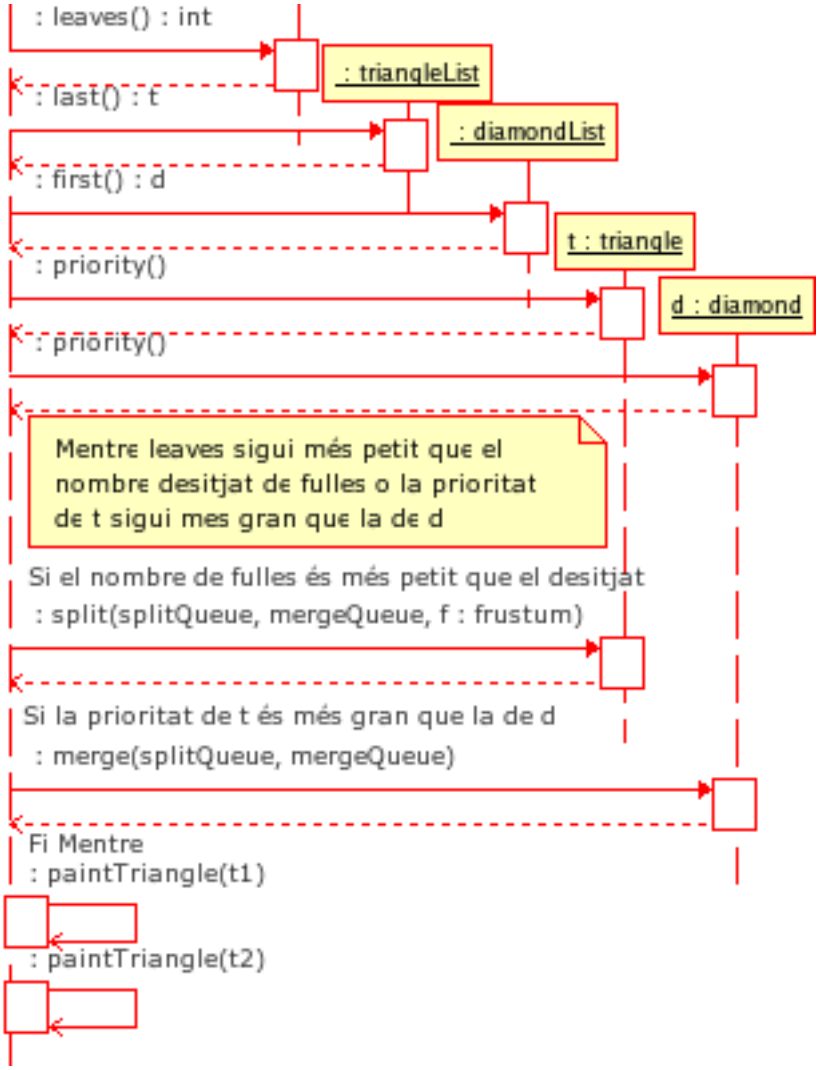

Figura 25: Diagrama de seqüència - ROAM - paint (continuació)

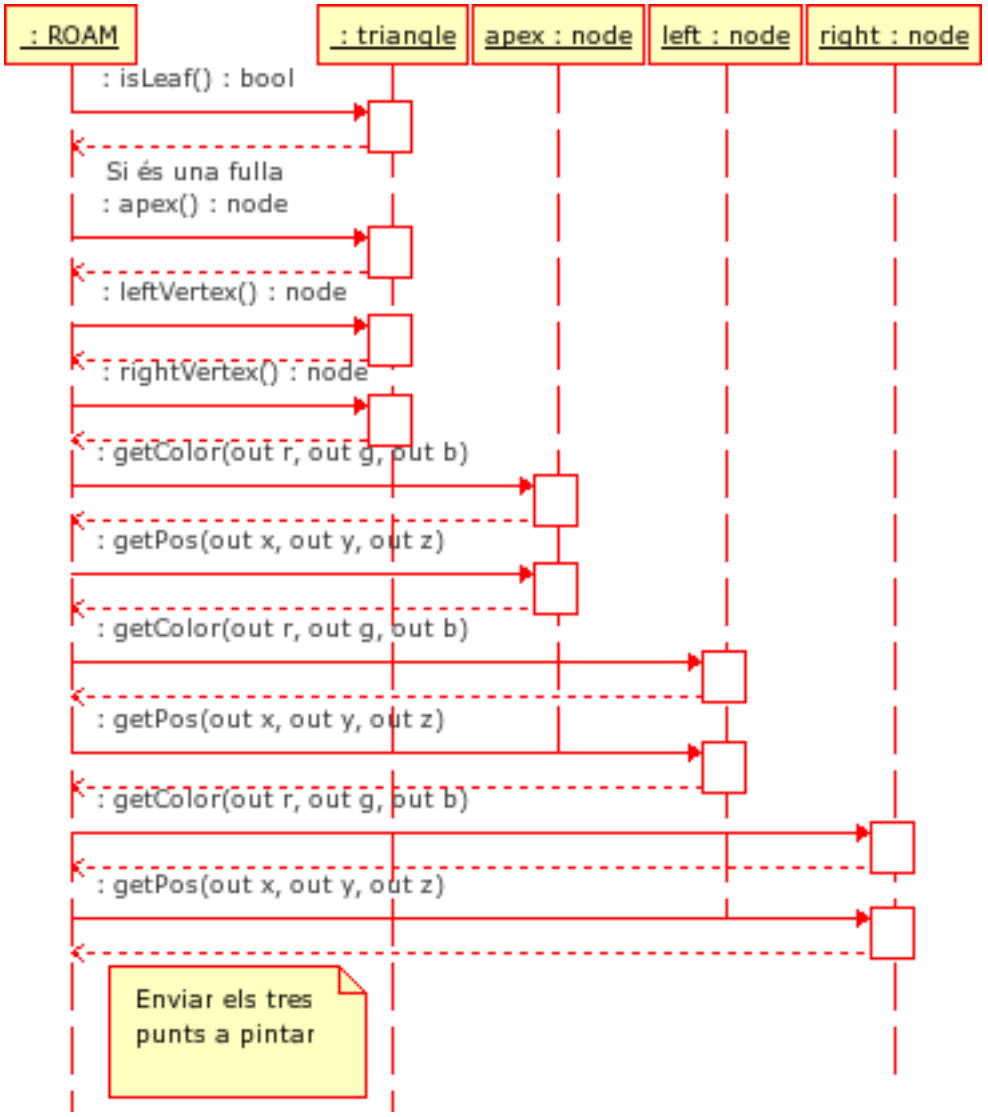

Figura 26: Diagrama de seqüència - ROAM - paintTriangle

Figura 27: Diagrama de seqüència - ROAM - paintTriangle (continuació)

5.6 Escala de colors

A l'hora de visualitzar el terreny s'ha decidit assignar-li a cada vèrtex un color que tingui a veure amb la seva alçada, d'aquesta manera és més fàcil per l'observador diferenciar les alçades de cada terreny. L'esquema que s'ha utilitzat és bastant simple, el punt més baix és de color blau, segons es va pujant d'alçada es va degradant el color cap al verd, de forma que els punts amb l'alçada mitja tenen color verd, la resta de punts a més alçada tenen un color que es va degradant del verd al vermell, que és el color dels punts més alts del terreny.

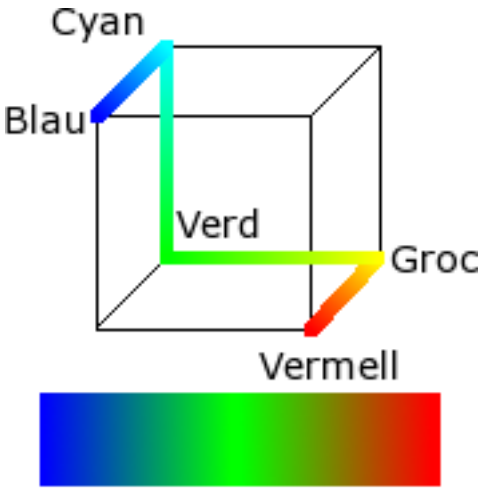

Figura 28: Escala de colors que s'ha utilitzat

6 Implementació

6.1 Requeriments no funcionals

En aquest capítol explicarem les decisions d'implementació que s'han pres per garantir els requeriments no funcionals mencionats a l'apartat 4.3

- **Traduïbilitat:** L'anglès és l'idioma en el qual s'han escrit els textos de la interfície original, ja que com és l'idioma més extés, és més fàcil trobar traductors per a la resta d'idiomes, en canvi si els textos estiguessin originalment en català o castellà la quantitat de traductors es reduiria de forma considerable. Tots els textos de l'aplicació estan a la part de r3v i no a la part de la llibreria libroam, això ha permés utilitzar les facilitats proporcionades per Qt per la traducció de programes. S'han utilitzat els programes lupdate, lrelease i linguist i la funció tr() que marca els textos que han de ser traduïts i els tradueix en cas que existeixi la traducció d'aquesta cadena a l'idioma que s'està utilitzant.

  - lupdate és el programa que s'encarrega de llegir els fitxers del codi font, buscar els textos que han de ser traduïts i actualitzar fitxers .ts que és on els traductors han de fer les traduccions.
  - linguist és el programa que han d'utilitzar els traductors per escriure les traduccions de cadascuna de les cadenes de text que s'han de traduir.
  - lrelease és el programa que transforma els fitxers .ts en fitxers .qm. Els fitxers .qm són representacions binàries de la traducció que utilitza el programa un cop s'està executant.

  Cada cop que s'executa el programa r3v en un ordinador es consulta la variable *QLocale::system().name()* que dóna el nom de l'idioma que s'està usant en el sistema operatiu, amb aquest nom es cerquen les traduccions disponibles, si n'hi ha alguna que concorda es fa servir aquesta, si no hi ha traducció a aquest idioma es fan servir els textos en anglès.

- **Portabilitat:** Per aconseguir que tant r3v com libroam es puguin compilar, i per tant, executar a la majoria de sistemes disponibles s'ha pres la decisió d'utilitzar C++ i la seva llibreria estàndard per a la programació, ja que C++ és un llenguatge pel qual hi ha compiladors disponibles per la majoria de sistemes operatius. D'igual manera la interfície està feta en Qt, un toolkit

en C++ que permet crear interfícies gràfiques d'usuari i té versions per a Linux/Unix, Mac OS i Windows. Finalment hi ha el problema de portabilitat de l'API 3D, això s'ha ressolt utilitzant OpenGL, una API 3D que també té implementacions per a la majoria de sistemes operatius actuals. Per conèixer més a fons que és una API 3D podeu consultar l'annex 12.2

- **Independència entre la interfície i l'algorisme:** Per aconseguir que la implementació de l'algorisme sigui independent de la interfície gràfica, s'ha programat l'algorisme com a una llibreria que utilitza tan sols C++ i les classes que proporciona la llibreria estàndard de C++, d'aquesta manera el canviar de Qt a un altre sistema per crear interfícies gràfiques com Gtk només suposa reescriure la classe que s'ocupa de gestionar els esdeveniments de teclat i ratolí, tota la resta és completament independent.

- **Programari lliure:** Per tal de poder garantir que el programa resultant pugui tenir llicència GPL, a part de declarar que el nostre codi és GPL també cal que totes les llibreries que s'usin siguin GPL. Qt que és l'única llibreria usada ho és.

6.2 Obtenció d'altures de coordenades no existents

Un dels problemes que s'han hagut de solucionar a l'hora de fer la implementació de l'algorisme és l'obtenció de les altures de coordenades no existents al mapa, és a dir, coordenades que no tenen una alçada associada al mapa. Això és un problema ja que l'algorisme pot demanar l'altura de qualsevol punt del mapa, mentre que els fitxers només proporcionen altures per un nombre finit de punts. És evident, per tant, que la resta de coordenades ens les hem *d'inventar*. El problema rau en com crear aquestes noves altures, ja que si donem una altura que no es correspon a aquell punt estarem enganyant a l'algorisme i fent que comenci a partir aquella zona quan potser en la realitat no cal.

Hi ha moltes maneres d'aproximar alçades per a una coordenada donada sabent l'alçada dels punts del voltant, però moltes d'elles utilitzen costoses funcions matemàtiques que no ens podem permetre implementar en aquest programa, ja que és molt important ser eficient a l'hora de demanar l'alçada d'un punt del mapa. Podem veure una explicació de la solució utilitzada a la figura 29, per implementar aquesta solució només calen sumes, restes i alguna multiplicació.

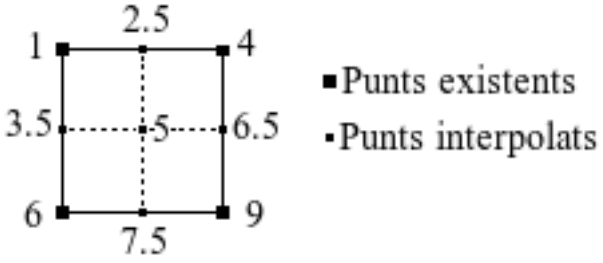

Figura 29: Obtenció d'altures de coordenades no existents

Com es pot veure a la figura s'ha utilitzat una solució que consisteix en emprar quatre punts que coneixem per interpolar l'alçada de cinc punts addicionals, quatre d'ells sobre les línies que uneixen els punts, essent el valor assignat la mitjana de les alçades dels dos punts, i el cinquè, el punt situat en el centre dels quatre punts, essent l'alçada interpolada la mitjana de tots quatre. A partir d'aquí podem veure que ja coneixem quatre punts per formar quatre quadrants més, per tant podem continuar subdividint el mapa fins a obtenir la coordenada que més ens interessi.

La solució implementada no és exactament aquesta, ja que seria massa costosa per la quantitat de recursions que caldria fer per segons quins punts. El mètode emprar es basa en la transformació a fórmula matemàtica de l'esperit d'aquest mètode, per tant, amb un sol pas podem obtenir l'alçada interpolada de qualsevol punt.

6.3 Generació de Makefiles

Qt té la seva pròpia eina per la generació automàtica de Makefiles anomenada qmake, però com ja s'ha comentat no volem dependre de Qt, per tant, hi havia dos opcions, la creació manual dels Makefiles o usar un generador automàtic de Makefiles que sigui *neutral*. La creació manual de Makefiles va quedar descartada ràpidament, ja que era una opció més pesada i menys potent, per tant s'havia de decidir quin generador automàtic usar. Aquesta decisió tampoc va ser gaire complicada, ja que encara que hi ha vàries eines que poden fer aquesta tasca, autotools és la més comuna i es pot trobar a quasi totes les distribucions de Linux i també funciona a quasi tots els sistemes Unix.

6.4 Autotools

Encara que sovint es parla d'Autotools (tal i com es farà en aquest document), Autotools és en realitat un paraula utilitzada per referir-se a un conjunt d'eines el nom de les quals comença per auto-, com autoheader, automake i autoconf. La sintaxi dels fitxers utilitzats per les autotools pot semblar complicada inicialment però si ens limitem a fer coses simples com les que fa aquest projecte (compilar una llibreria i un executable que depèn d'ella) és força potent i fàcil de fer servir.

Autoconf és l'eina que serveix per comprovar que la plataforma en la que es compilarà el programa té les eines i llibreries que calen. A part, també serveix per detectar particularitats de la plataforma, per exemple si en aquesta plataforma una funció té uns paràmetres no estàndard, es poden definir uns símbols que més tard farem servir en el codi font per adaptar-nos a aquestes particularitats. Per part del programador només s'ha de crear el fitxer configure.in que conté paraules clau que indiquen totes les comprovacions que volem fer, a partir d'aquest fitxer, autoconf genera un script anomenat configure que en executar-se en l'ordinador que està compilant el programa fa les comprovacions i avisa en cas de que alguna condició no es compleixi.

Per aquest projecte el fitxer configure.in es limita a comprovar l'existència d'un compilador de C++, que l'eina libtool es trobi instal·lada al sistema i finalment fa la comprovació de si Qt està instal·lat. En el cas de Qt es comprova si la versió instal·lada és 3.x o 4.0 i es defineix el símbol Qt4 de forma que el programa funcioni tant amb les versions 3.x com amb la versió 4.0 (de la qual, encara no hi ha versió final, però sí versions preliminars suficientment funcionals per compilar aquest projecte).

Automake és l'eina que serveix per generar els Makefile. De forma ràpida un Makefile és un fitxer que indica com s'ha de compilar un programa i la depèndencia dels fitxers entre si, de forma que en canviar un fitxer es tornin a recompilar només aquells que hi tenen alguna relació. Aquesta eina genera fitxers Makefile.in a partir de fitxers Makefile.am, els fitxers Makefile.in es converteixen en fitxers Makefile al executar-se el configure creat per Autoconf. Als fitxers Makefile.am només cal indicar quines són les llibreries i els fitxers de codi font que utilitza el nostre programa/libreria i Automake s'encarrega de generar un Makefile complet amb els típics install, clean, uninstall, etc.

Per aquest projecte hem utilitzat fitxers Makefile.am per generar la llibreria libroam i per generar l'executable r3v, les opcions usades han estat bastant simples, al fitxer Makefile.am de libroam s'ha indicat que volem que sigui una llibreria estàtica generada per libtool formada per tots els fitxers .cpp del directori i a més a més usant les llibreries GL i GLU. Per a l'executable r3v, s'ha indicat que es volia generar un executable usant el codi de la interfície en Qt i usant la llibreria libroam així com la llibreria Qt.

L'ús de les autotools presenta un petit problema en quant a la portabilitat del projecte, ja que no hi ha versió nativa de les autotools per a Windows. Això significa que el projecte no pot ser compilat directament sota Windows, cal adaptar-lo a les eines del compilador de C++ que s'usi; un petit problema de portabilitat però que no afecta de cap manera que el codi del programa continuï sent portable.

6.5 Lector de format DEM

El format DEM dóna moltes més dades de les necessàries per implementar un visualitzador de superfícies, com ara, nom , descripció i coordenades del mapa, estadístiques sobre l'exactitud dels valors, etc. Aquestes dades poden ser molt interessants, però no en el nostre context, per tant, s'ha decidit que la implementació del lector de fitxers de format DEM per a crear un objecte de la classe r3vMap, ignori tots aquest valors i es centri en les dades d'altitud.

7 Captures de pantalla

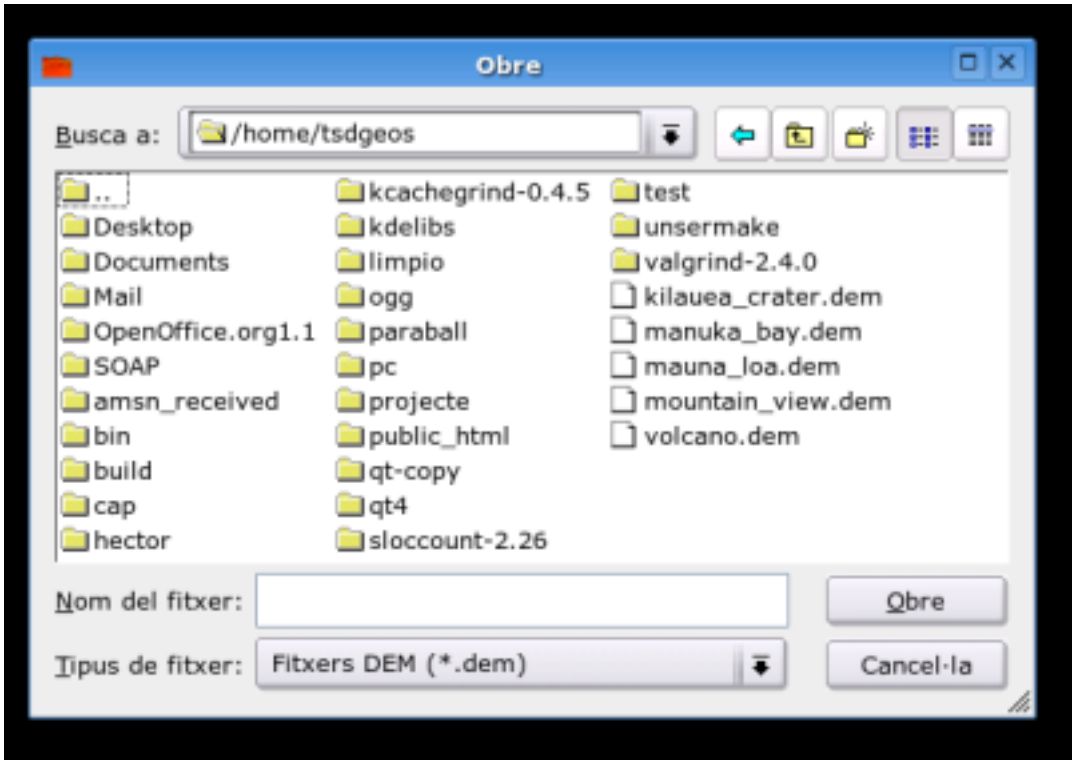

Figura 30: Pantalla d'obertura d'un mapa

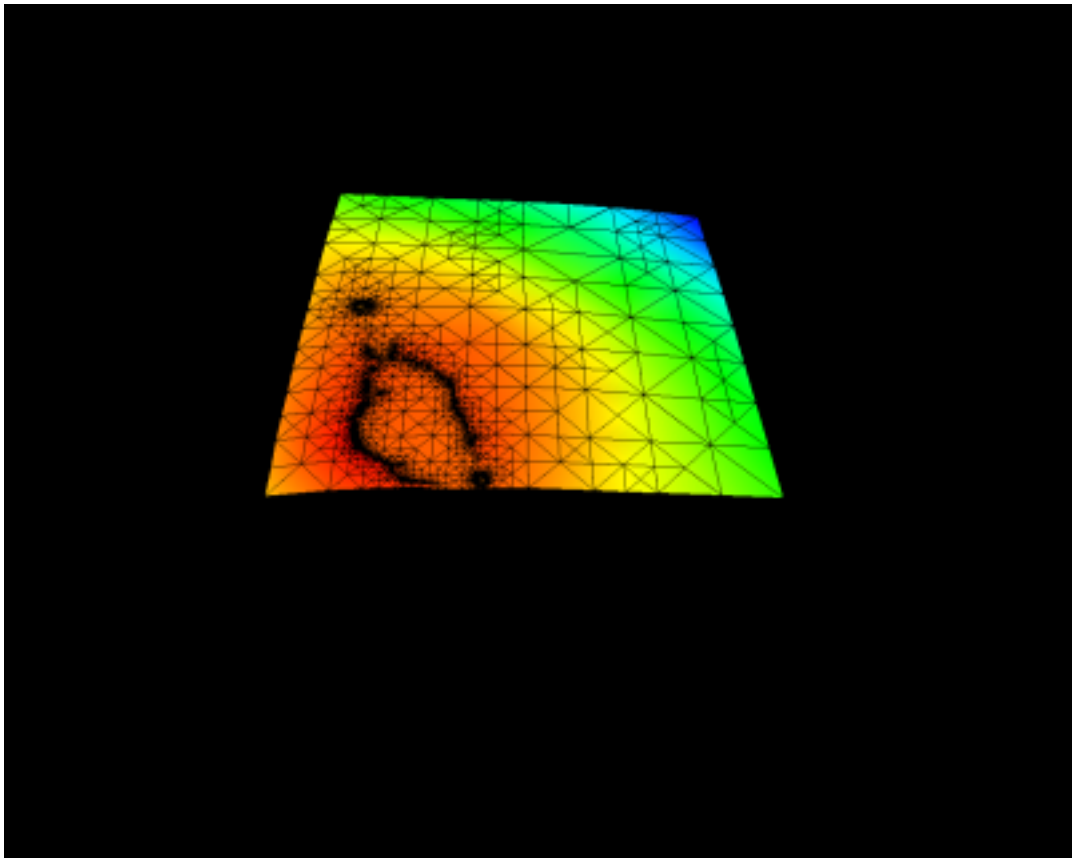

Figura 31: Vista general d'un mapa

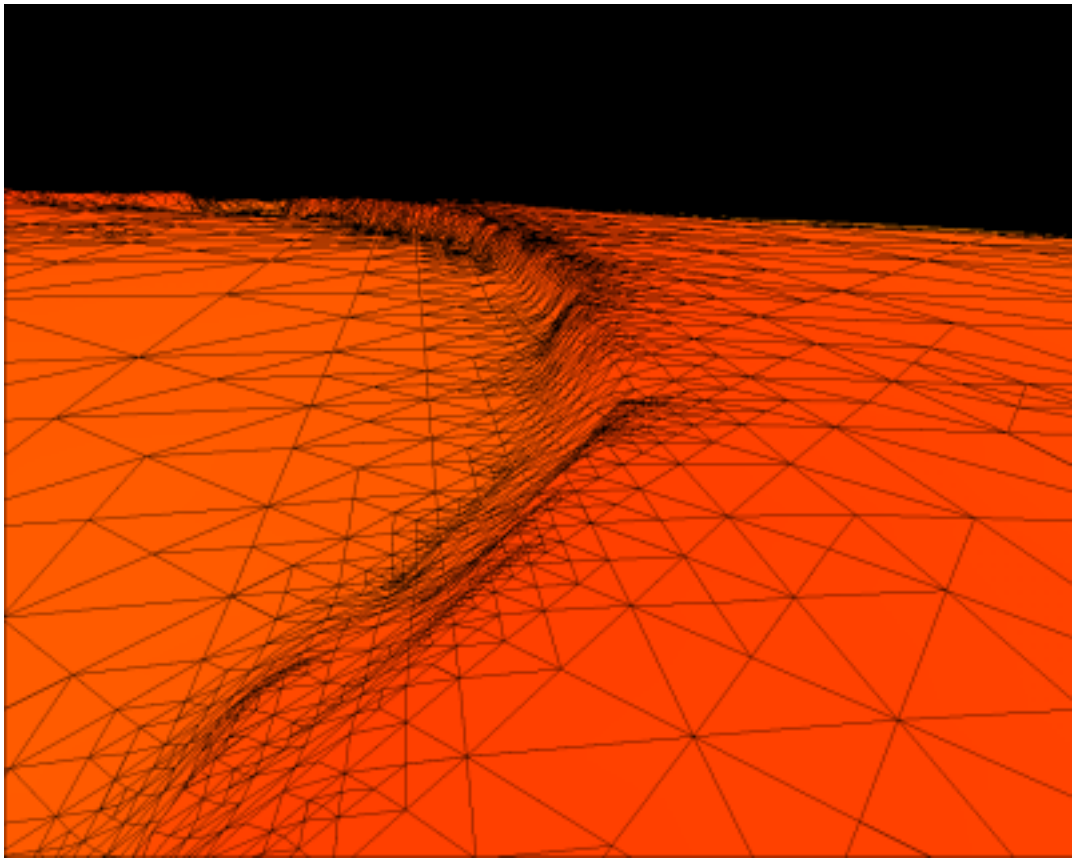

Figura 32: Vista detallada del mapa anterior

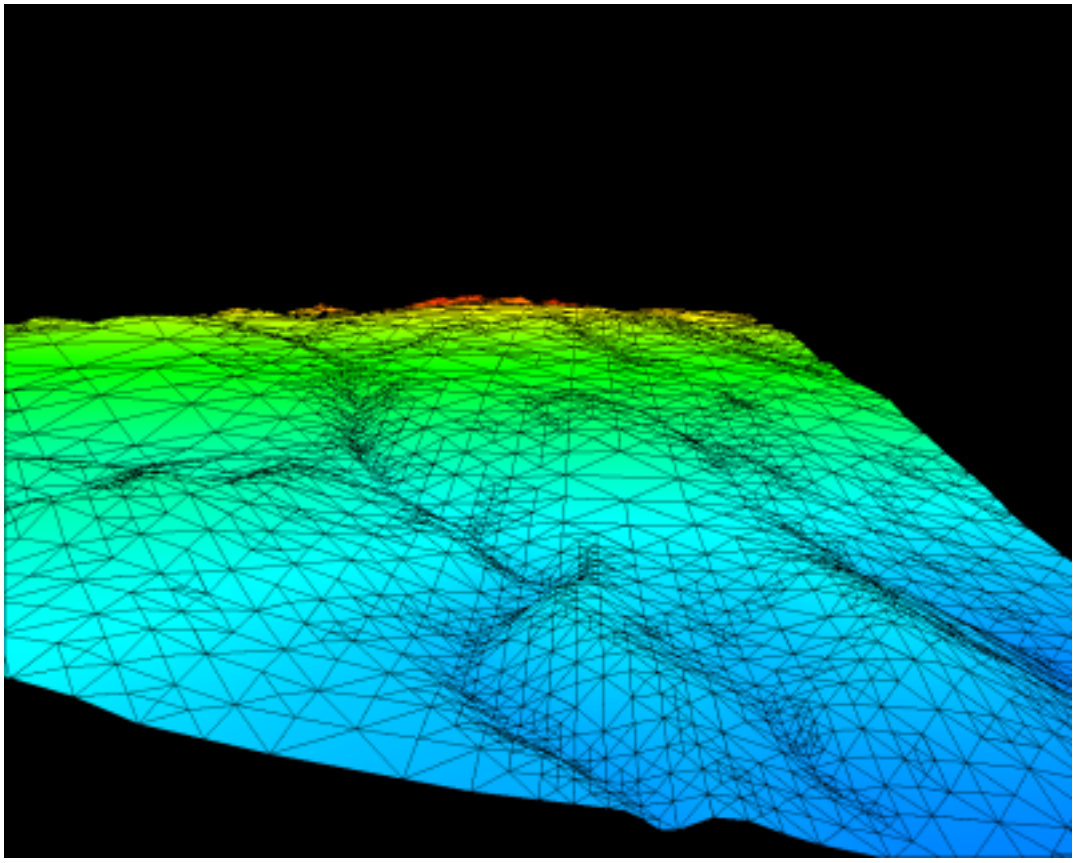

Figura 33: Vista detallada d'un altre mapa

8 Recursos i Anàlisi de costos

8.1 Recursos utilitzats

8.1.1 Recursos humans

Els recursos humans per la realització d'aquest projecte s'han reduït a mi mateix, ja que aquestes són les normes de la realització de projectes a la FIB. En alguns casos això no és totalment òptim, per exemple per la redacció de textos, a una empresa no seria un Enginyer Informàtic qui s'ocuparia, però com s'ha dit el projecte és un treball per una sola persona.

En cas d'haver de buscar a vàries persones per desenvolupar el projecte hauria estat necessari buscar a un dissenyador amb experiència en el món dels gràfics 3D, un programador amb experiència amb OpenGL, C++ i Qt i un escriptor de documentació.

La dedicació en temps ha estat variable al llarg del temps de la realització del projecte (que s'ha fet entre Febrer i Juny de 2005) però es pot situar en torn a una mitja de 20 hores setmanals.

8.1.2 Recursos tecnològics

En quant als recursos tecnològics utilitzats tenim dues branques que es poden diferenciar fàcilment, els recursos hardware i els software.

Els recursos software utilitzats per a la realització del projecte han estat tots programes lliures, en especial tots ells part de la distribució Suse Linux 9.1 (utilitzada a la FIB) o de la distribució Mandriva 2005 (utilitzada a casa). Els programes essencials per a la realització d'aquest projecte han estat:

- ***gcc:*** Compilador utilitzat, concretament g++, el compilador de C++ del programari gcc
- ***kate:*** Editor de text
- ***latex:*** Sistema de creació de documents
- ***kile:*** Editor de text especialitzat en latex
- ***gimp:*** Editor d'imatges utilitzat per la creació d'alguns gràfics explicatius

- ***umbrello:*** Programa CASE[1] utilitzat per fer els diagrames de classes i diagrames de seqüència

Els recursos hardware utilitzats es poden resumir en:

- PCs de desenvolupament: Evidentment tenir ordinadors on poder desenvolupar i provar el codi és molt important, en aquest cas se n'han fet servir dos de diferents. Un a la FIB i un altre a casa. En tots els casos és important poder disposar de diferents màquines on fer les proves per veure que el funcionament és correcte a totes elles, però en el cas d'una aplicació 3D és una mica més important, ja que a vegades el comportament de les targetes 3D amb les mateixes comandes OpengGL varia de xip a xip, i per tant quantes més targetes es provin milor. El PC de la FIB té les següents característiques
  - Processador: Intel Pentium 4 3.00GHz
  - Memòria: 512 MB
  - Disc Dur: 20 GB
  - Targeta gràfica: Ati Radeon 9200

  mentre que el PC de casa té aquestes
  - Processador: AMD Athlon XP 1600+
  - Memòria: 384 MB
  - Disc Dur: 160 GB
  - Targeta gràfica: PowerVR Kyro 2

  Cal notar que a la FIB el disc dur utilitzat és un disc dur extraïble de forma que es pot utilitzar a qualsevol dels PCs disponibles de la sala de projectistes.
- Connexió a la xarxa: En el món de la programació actual tenir connexió a internet és molt important ja que és una font de consulta de manuals molt important. A la FIB s'ha utilitzat la connexió LAN de la facultat mentre que ha casa s'ha fet servir una connexió via mòdem 56K.

[1] Computer-aided software engineering

8.1.3 Recursos físics

Una part que normalment s'oblida al fer aquest tipus de resum dels recursos utilitzats per fer la realització d'un projecte són els recursos físics, és a dir, cadires, taules, edificis, etc. Tot això és completament necessari per poder dur a terme qualsevol projecte, i, de vegades pot ser un cost bastant important, com per exemple en el cas del lloguer o compra d'una oficina. En el cas del projecte que ens ocupa no s'ha hagut d'adquirir cap dels recursos físics utilitzats ja que la Facultat ja proporciona lloc on desenvolupar el PFC.

8.2 Anàlisi de costos

En aquesta taula podem trobar el tipus de personal que cal per realitzar cadascuna de les tasques de les que s'ha composat la realització del projecte això com el temps que s'ha trigat en fer cadascuna d'elles.

Tasca	**Tipus**	**Hores de realització**
Planificació inicial del projecte	Analista	15 hores
Especificació	Analista	75 hores
Disseny	Analista	75 hores
Implementació	Programador	125 hores
Realització de proves	Provador de software	25 hores
Redacció de la memòria	Documentador	100 hores
Total		**415 hores**

Cada un dels tipus de personal té un sou diferent degut a que els coneixements no són els mateixos. A la taula següent hi figura el cost hora que tindria cada tipus de perfil realitzat.

Tipus de personal	**Cost**
Analista	40 €/hora
Programador	30 €/hora
Provador	25 €/hora
Escriptor de documentació	35 €/hora

Això ens dona un cost total per pagar al personal de:

Tasca	**Cost**
Planificació inicial del projecte	15 * 40 = 600 €
Especificació	75 * 40 = 3000 €
Disseny	75 * 40 = 3000 €
Implementació	125 * 30 = 3750 €
Realització de proves	25 * 25 = 625 €
Redacció de la memòria	100 * 35 = 3500 €
Total	**14475 €**

A aquests 14475 € caldria sumar el cost dels recursos hardware, software i físics, però això és complicat de comptar, ja que en el cas de hardware i software el material necessari per la realització del projecte no seria exclusiu per aquest projecte en cas

que es tractés d'una empresa, per tant no seria lògic assignar el cost total del material al cost d'aquest projecte. També es pot dir el mateix del cost dels recursos físics, ja que no es pot assignar completament el cost del lloguer d'oficines, mobiliari, etc. que seria necessari per dur a terme la realització del projecte. Per tant, fent una aproximació dels costos d'amortització que es podrien assignar a cadascun dels recursos anteriors es podria xifrar en torn dels 20000€ el cost total de la realització del projecte.

9 Conclusions i Treball futur

El resultat de la realització d'aquest projecte final de carrera ha estat bastant satisfactori en termes d'objectius assolits, tant en la part de requeriments que es volia que complís l'aplicació com en la part de formació final de l'alumne.

En quant al compliment de requeriments funcionals i no funcionals que es van establir en la reunió inicial que vam tenir el projectista i el director, es pot dir que s'han assolit totalment, ja que no hi ha cap requeriment dels que vam parlar inicialment que no hagi acabat estant present a l'aplicació final.

La part de formació final de l'alumne que se li suposa a la realització d'un projecte d'aquesta envergadura també ha estat profitosa ja que he adquirit més coneixements sobre OpenGL i C++ en general dels que ja tenia, així com també m'he enfrontat a la tasca de gestionar un projecte tan llarg per primera vegada, aprenent així totes les coses que això implica, com gestionar el temps d'una manera eficient.

Encara que s'hagin complert tots els requeriments que es van parlar en la reunió inicial per definir aquest projecte això no vol dir que no hi hagi coses a millorar o ampliar per tal que libroam o r3v siguin millors. Aquesta és una llista de les vàries tasques que es poden afrontar com a treball futur:

- Organitzar els triangles en strips: Organitzar els triangles en strips no és una tasca senzilla, però fer-ho pot reportar guanys de rendiment, ja que al utilitzar strips de triangles el nombre de vèrtexs que s'envien a pintar es redueix considerablement, i a més a més, algunes targetes gràfiques tenen optimitzacions especials per gestionar els strips de triangles.

- Càlcul de prioritat diferida: L'observador, normalment, no es mou bruscament de lloc ni gira exageradament, per tant, realment no caldria recalcular la prioritat de tota la llista de diamants i triangles per cada nova posició de l'observador, ja que aquesta prioritat tampoc variarà de forma exagerada, per tant, recalcular només un tant per cent donat dels elements d'aquestes llistes pot ajudar a optimitzar l'algorisme, evidentment s'haurien de programar comprovacions que asseguressin que l'observador no s'ha mogut exageradament, ja que en aquest cas si seria necessari recalcular tota la llista de prioritats.

- Aturada del càlcul de prioritat per temps: Una altra característica que es pot afegir a l'algorisme ROAM és donar un límit de temps que volem que es trigui en obtenir la malla de triangles per la nova posició. Això es pot implementar fàcilment, ja que només cal posar la comprovació de temps al *while* que va regenerant l'arbre de triangle fent les fusions i particions. El fet d'aturar la regeneració de l'arbre de triangles suposarà que no tindrem la malla òptima per la posició, però el que si serà és la millor malla possible en el temps donat, ja que la regeneració es va fent fusionant els pitjors diamants i partint els millors triangles. Aquesta característica encara que és fàcil de fer, no s'ha implementat ja que als ordinadors disponibles a l'actualitat la part de regeneració de la malla de triangles no és costosa, sent el pintat per pantalla dels triangles la part que triga més temps.

10 Bibliografia

- Paper sobre l'algorisme ROAM
  http://www.llnl.gov/graphics/ROAM/roam.pdf
- Especificació del format DEM
  http://rockyweb.cr.usgs.gov/nmpstds/acrodocs/dem/2DEM0198.PDF
- Documentació de Qt
  http://doc.trolltech.com/3.3
- Documentació de libstdc++
  http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html
- Documentació de la STL de SGI
  http://www.sgi.com/tech/stl/table_of_contents.html
- Document en quant a view frustum culling
  http://www.racer.nl/reference/vfc.htm
- Document en quant a view frustum culling
  http://www.sjbaker.org/steve/omniv/frustcull.html
- Mapes de Hawaii en format DEM
  http://duff.geology.washington.edu/data/raster/tenmeter/hawaii/

11 Índex de figures

Índex de figures

12 Annex

12.1 Gràfics per computador

Els gràfics per computador és un camp de la computació en el qual s'utilitzen els ordinadors per generar imatges, ja siguin completament sintètiques o generades a partir d'informació obtinguda del món real.

Els gràfics per computador tenen dos grans camps, els gràfics 2D i els gràfics 3D.

12.1.1 Gràfics 2D

Els gràfics 2D són aquells que representen imatges bidimensionals en pantalla. Hi ha dos tipus bàsics d'imatges 2D, els mapes de bits i les imatges vectorials.

Els mapes de bits són bàsicament una graella de píxels bidimensional. Cada píxel té una posició assignada dintre de la graella i emmagatzema el seu color. Un mapa de bits té una resolució finita determinada pel nombre de columnes i files, i per tant si es vol mostrar més gran del que realment és, sol aparèixer l'efecte de pixelació consistent en que l'observador és capaç de distingir cada un dels píxels de forma separada, cosa que no passa quan la imatge es visualitza amb la seva mida real.

Les imatges vectorials tenen les dades geomètriques de forma precisa. Els punts es donen en un sistema de coordenades, les dades de la imatge vectorial també contenen la descripció de les connexions entre els punts(per formar línies o rutes), el color de cada punt o línia, el gruix de cada línia o secció, etc. La majoria de sistemes gràfics vectorials també tenen primitives pels tipus de formes més bàsics com els cercles o rectangles. Aquestes dades permeten generar la imatge des de zero. El fet que la imatge estigui guardada amb una descripció de com s'ha de *crear*, permet que les imatges vectorials no sofreixin mai de l'efecte de pixelació encara que també les fa més lentes de visualitzar. Per mostrar per pantalla una imatge vectorial s'ha d'acabar creant un mapa de bits.

12.1.2 Gràfics 3D

Els gràfics 3D tenen dos branques bastant diferenciades, els gràfics 3D per ser visualitzats en una pantalla (ja sigui d'ordinador, televisió, etc.) i els gràfics tridimensionals que es representen de forma volumètrica usant diferents aparells. En

aquest document ens centrarem en explicar els gràfics 3D per ser visualitzats en pantalla ja que els altres estan menys desenvolupats i a més a més no tenen cap relació amb l'objectiu del projecte.

Els gràfics 3D es diferencien dels gràfics 2D principalment pel fet que en els gràfics 3D hi ha una representació virtual tridimensional dels objectes de l'escena que s'utilitza per fer els càlculs i per obtenir la imatge 2D que finalment s'acabarà representant en pantalla mentre que en els gràfics 2D o bé no hi ha objectes (mapa de bits) o els objectes són bidimensionals (imatges vectorials). En general, els gràfics 3D es poden comparar amb una fotografia d'una escultura, mentre que els gràfics 2D es correspondrien amb un dibuix.

12.1.3 Gràfics interactius vs Gràfics realistes

Dins dels gràfics generats per computador (ja siguin 2D o 3D) hi ha dos grans branques. Els gràfics interactius i els gràfics realistes.

Els gràfics interactius són aquells en els que l'observador pot influir en la generació dels gràfics. És a dir, l'usuari del programa pot rotar, moure, escalar, etc. els objectes de l'escena a voluntat. Això és útil per a jocs, simuladors i visualitzadors de dades en general. La característica principal que es demana a aquests gràfics és la velocitat de generació, que ha d'estar al voltant de les 25 actualitzacions per segons, de forma que l'observador vegi una suavitat en el moviment, per sota de les 15 actualitzacions per segon el moviment comença a semblar que vagi a salts. Per aquesta raó també se'ls sol anomenar gràfics en temps real. Degut a que la velocitat és el principal factor en aquests tipus de gràfics la qualitat no és molt gran, encara que gràcies a que les targetes acceleradores estan evolucionant molt ràpid, la qualitat ha pujat molt en poc temps. La base d'aquest tipus dels gràfics en el món 3D són els triangles, tot és genera a partir d'una acumulació de triangles, fins i tot els cercles.

Els gràfics realistes són aquells on el que prima és que la imatge generada s'asembli el més possible a una imatge real. El destí principal d'aquest tipus de gràfics són les pel·lícules, anuncis, etc. que gràcies als gràfics realistes poden incloure imatges que abans serien o molt cares de fer o, fins i tot, impossibles de rodar. Aquest fet condiciona la velocitat de generació de les imatges i de vegades es triga hores en aconseguir una sola imatge estàtica, per tant, aquest mètode encara que aconsegueix imatges d'una qualitat espectacular no serveix per fer gràfics interactius. A

la generació de gràfics 3D s'usen mètodes que utilitzen fórmules matemàtiques que intenten modelar el comportament real de la llum.

12.2 API 3D

Una API (**A**pplication **p**rogramming **i**nterface) és un conjunt de definicions de com un component de software interactua amb un altre. És un mètode per aconseguir abstracció entre capes diferents de software, ja sigui entre capes de diferents nivells d'abstracció com de capes del mateix nivell d'abstracció que es dediquen a tasques diferents.

Una API 3D és una API que defineix les funcions necessàries per crear gràfics tridimensionals. Avui en dia hi ha dos APIs 3D dominants per la generació de gràfics interactius: OpenGL i Direct3D. Totes les APIs 3D per generar gràfics interactius serveixen també per generar gràfics 2D interactius ja que només cal fer que tots els objectes 3D es situïn en un mateix pla per fer que el món 3D sigui bidimensional.

12.2.1 OpenGL

OpenGL (**Open G**raphics **L**ibrary) és una especificació d'una API d'un llenguatge per escriure aplicacions que produeixen gràfics 3D. OpenGL és un estàndard definit per la *OpenGL Architecture Review Board*, formada per molts membres de la indústria dels computadors, ja siguin constructors d'ordinadors, de sistemes operatius o de hardware gràfic. Alguns dels membres són: Apple, ATI Technologies, Dell, Hewlett-Packard, IBM, Intel, Nvidia, SGI, etc. cosa que converteix OpenGL en un estàndard realment neutral i amb suport per la gran majoria de les plataformes gràfiques.

OpenGL va ser introduït el 1992 com a evolució de la interfície 3D IRIS GL de SGI. Un dels problemes de IRIS GL era que tot el que implementava la llibrera havia d'estar suportat pel hardware i per tant si el hardware no suportava una característica que es feia servir al programa, aquest no es podia executar. OpenGL va solucionar aquest problema proporcionant suport software per a les característiques que el hardware no tenia, cosa que en aquell moment va permetre executar programes 3D en sistemes no tan cars com els que fabricava SGI.

Les característiques bàsiques d'OpenGL són:

- **Multillenguatge:** Hi ha llibreries que permeten utilitzar OpenGL des de C, C++, Fortran, Ada, Java, Perl, Python, Visual Basic, etc. Per tant, es pot reutilitzar el codi OpenGL d'una aplicació en un llenguatge per programar la mateixa aplicació o similar en un llenguatge diferent.
- **Multiplataforma:** OpenGL està disponible per a Windows, Mac OS, Unix comercials, Linux i d'altres sistemes operatius menors, per tant una aplicació que utilitzi OpenGL no està limitada a executar-se en un sistema operatiu en particular.
- **Estable:** L'API d'OpenGL s'ha mantingut bastant estable durant més de 7 anys. Això permet que els programes continuïn funcionant sense problemes amb noves versions i a més a més facilita la feina dels programadors que no han d'anar modificant els seus coneixements cada poc temps.

12.2.2 Direct3D

Direct3D és la part que s'encarrega de la visualització tridimensional de l'API DirectX de Microsoft. Només hi ha versions de DirectX (i per tant de Direct3D) per als sistemes operatius de Microsoft (Windows 95 en endavant) encara que la consola XBox (produïda per Microsoft) té una API molt similar a la de DirectX.

Hi ha moltes versions de DirectX, cadascuna afegeix noves característiques sobre l'anterior. Això és útil en un món que es mou tan ràpidament com el dels gràfics per computador però també suposa que els programadors han d'anar readaptant el seu coneixement sobre la llibreria per poder programar per les últimes versions.

12.2.3 Primitives gràfiques 3D

La majoria d'API per la generació de gràfics 3D interactius tenen un conjunt similar de primitives, és a dir, *d'objectes* que podem fer servir per construir una escena. En aquesta secció ens centrarem en les primitives que té OpenGL.

- **GL_POINTS** - Cada punt que es dona es converteix en un punt a l'espai.
- **GL_LINES** - Cada dos punts que es donen es converteixen en una línia.
- **GL_LINE_STRIP** - Donats i punts, es creen $i-1$ línies. Els punts que uneix una línia amb índex k són els k i $k+1$.

- **GL_LINE_LOOP** - Donats i punts, es creen i línies. Els punts que uneix una línia amb índex k són els k i $k+1$ per $0 < k < i-1$ i els i i el 0 per l'última línia ($k = i$).
- **GL_TRIANGLES** - Cada 3 punts que es donen es converteixen en un triangle.
- **GL_TRIANGLE_STRIP** - Donats i punts, ($i > 3$) es creen $i-2$ triangles. El triangle amb índex k està format pels punts amb índex k, $k+1$ i $k+2$.
- **GL_TRIANGLE_FAN** - Donats i punts, ($i >= 3$) es creen $i-2$ triangles. El triangle amb índex k està format pels punts amb índex 0, $k+1$ i $k+2$.
- **GL_QUADS** - Cada 4 punts que es donen es converteixen en un quadrilàter.
- **GL_QUAD_STRIP** - Els quatre primers vèrtexs formen un quadrilàter. Cada parell successiu de vèrtexs es combina amb el parell anterior per formar un altre quadrilàter.
- **GL_POLYGON** - Té el mateix comportament que **GL_LINE_LOOP**, però en comptes de crear línies crea un polígon.

Podem veure exemples de les 10 primitives a la figura 34

12.3 Paradigmes de navegació 3D

La navegació d'escenes 3D té tres paradigmes fonamentals, Walk, Fly i Examine. Tota aplicació 3D implementa un d'aquests paradigmes, encara que no n'ha d'implementar només un.

12.3.1 Paradigma walk

Al paradigma walk, l'usuari del programa pot moure la càmera i per tant navegar per l'escena. L'usuari sol estar representat per una entitat dins de l'escena anomenada avatar. L'avatar està sotmés a la gravetat i per tant només pot caminar per terrenys plans, pujar pendents poc pronunciades i xoca contra les parets. Els moviments típics de l'avatar són desplaçar-se en qualsevol de les quatre direccions (endavant, endarrera, esquerra i dreta) i gir de la càmera per enfocar cap a qualsevol direcció possible. En aquest paradigma la visió pot ser en primera persona o en tercera persona. A la visió primera persona, la càmera es situa als ulls de l'avatar, veient-se

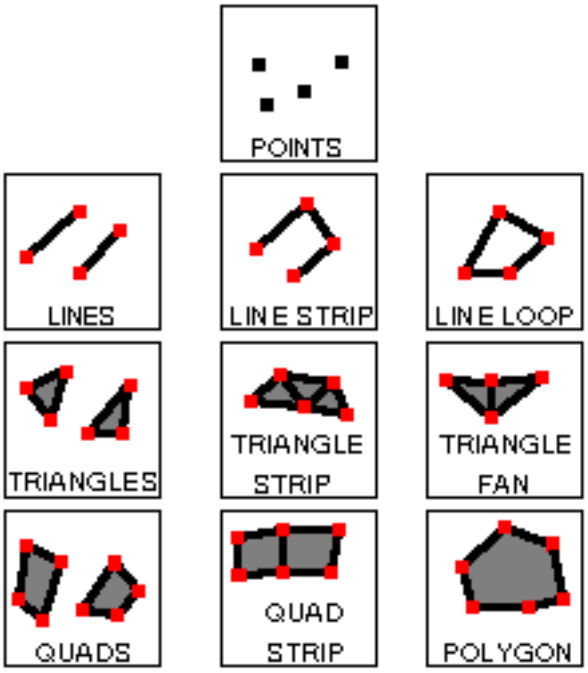

Figura 34: Primitives gràfiques d'OpenGL

allò que veuria l'avatar, per tant, de l'avatar només es sol veure les mans i peus. A la visió en tercera persona la càmera es situa fora de l'avatar, normalment per darrera d'ell, de forma que ens permet veure la situació d'una manera més global. Aquesta càmera té més problemes de gestió, ja que s'ha d'anar amb compte de que la càmera no atravesi parets quan l'avatar s'apropa a elles, ja que si ho fes no veuriem l'avatar, sinó la paret. A més a més la visió en primera persona sol donar més sensació d'inmersió, per tant la majoria de programes 3D que utilitzen el paradigma walk solen optar per ella.

12.3.2 Paradigma fly

El paradigma fly és gairebé igual al paradigma walk, sent la no existència de la gravetat la diferència més gran. El paradigma fly se sol utilitzar en simuladors espaials o bé en els mateixos escenaris que utilitzen el paradigma walk per donar-li a l'usuari més llibertat a l'hora de reconèixer l'escena, per exemple és molt possible que als dissenyadors de pantalles els interessi poder volar per l'escenari per així poder desplaçar-se entre diferents alçades de forma més ràpida. Així mateix a vegades el paradigma fly desactiva la detecció de col·lisions permetent que l'avatar atravesi les parets.

12.3.3 Paradigma examine

Al paradigma examine l'observador no es mou, sempre està a la mateixa posició i mira en la mateixa direcció. La interactivitat d'aquest paradigma és a la part dels objectes i no a la de l'observador. Normalment hi ha uns quants objectes (de vegades només un) al punt on està enfocant la càmera i l'usuari pot rotar, escalar, desplaçar els objectes de forma que pugui examinar l'objecte completament. Aquest paradigma es sol usar en aplicacions que permeten la visualització d'ossos, peces de maquinària, etc.

12.4 Arbres binaris de triangles

Un arbre binari de triangles és un arbre que serveix per organitzar triangles de forma jeràrquica. El triangle arrel $T = (v_a, v_0, v_1)$ és un triangle isòsceles amb àpex v_a. Els fills de l'arrel es defineixen partint l'arrel en dos parts creant una nova aresta que va des del vèrtex v_a al punt mig v_c de l'aresta base (aresta entre v_0 i v_1). El fill esquerre de T és $T_0 = (v_c, v_a, v_0)$ mentre que el fill dret és $T_1 = (v_c, v_1, v_a)$. La resta de l'estructura de l'arbre de triangles és una repetició recursiva del procés de partició explicat anteriorment. Dins de l'arbre cada triangle té un nivell l definit pel nombre de vegades que s'ha partit al triangle arrel per arribar a generar aquell triangle en concret. A la figura 35 podem veure una representació dels nivells 0 a 5 d'un arbre binari de triangles.

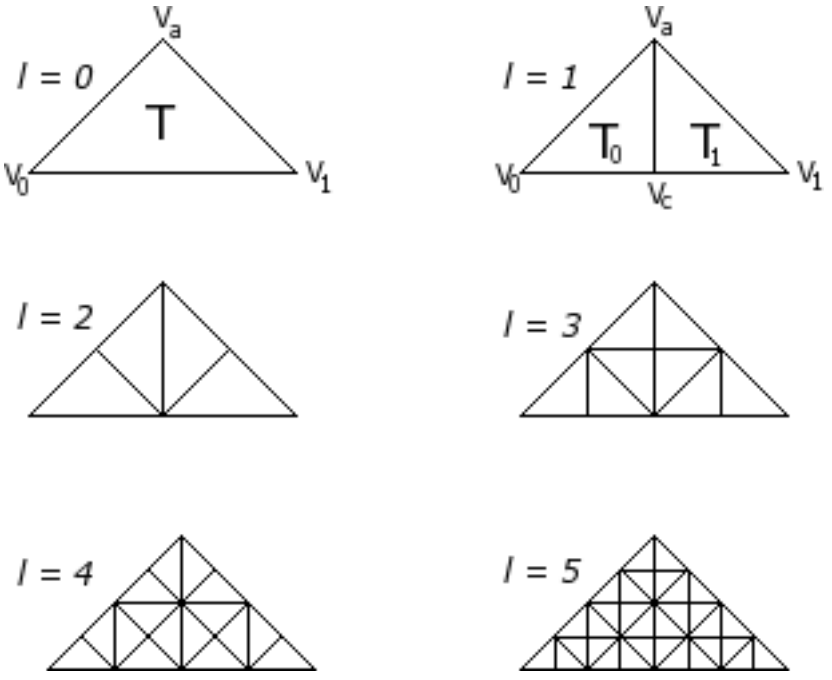

Figura 35: Representació dels nivells 0 a 5 d'un arbre binari de triangles

12.4.1 Representació de superfícies amb arbres binaris

Per representar una superfície amb arbres binaris de triangles es fan servir dos arbres de triangles situats de forma que comparteixin la seva aresta base, d'aquesta forma es cobreix una superfície quadrada (Figura 36). Un cop tenim una estructura que permet cobrir tot un quadrat només cal assignar a cada vèrtex una alçada $w(v)$ per tenir una forma eficient de guardar les alçades d'un terreny.

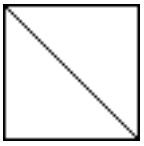

Figura 36: Dos arbres de triangles serveixen per representar una superfície quadrada

12.4.2 Conjunts d'arbres binaris de triangles

Una triangulació d'arbres binaris de triangles (triangulació d'ara en endavant) és un conjunt d'arbres binaris que tenen alguna relació de veïnatge entre ells, ja sigui compartint un vèrtex o una aresta. Un cas especial de triangulació és el diamant, triangulació formada per dos triangles que comparteixen l'aresta base i tenen el mateix nivell, la figura 36 és un exemple de diamant. Un fet important d'una triangulació és que donat un triangle, el triangle amb el que comparteix la base és del mateix nivell o del nivell superior. A la figura 37 podem veure un exemple de les dues situacions, el triangle A comparteix la base amb un triangle del seu nivell mentre que el triangle B comparteix la base amb un triangle d'un nivell superior (l més petita).

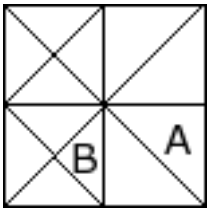

Figura 37: Possibilitats de compartir la base

12.4.3 Operacions sobre diamants

Sobre un diamant es poden realitzar dues operacions, split i merge. Split (partir) consisteix en crear els fills de T i del seu triangle base T_B de forma que s'obté una triangulació formada pels triangles T_0, T_1, T_{B0} i T_{B1}. Merge (fusionar) consisteix en tornar als triangles T i T_B a partir dels seus quatre fills. Per tant, definirem un diamant fusionable com aquell diamant els triangles del qual tenen fills però aquests fills no en tenen. Es pot veure una representació dels processos de partició i unió a la figura 38.

Només els triangles que tenen el mateix nivell que el triangle amb el que comparteixen la base són susceptibles de ser partits. Aquells que comparteixen la base amb

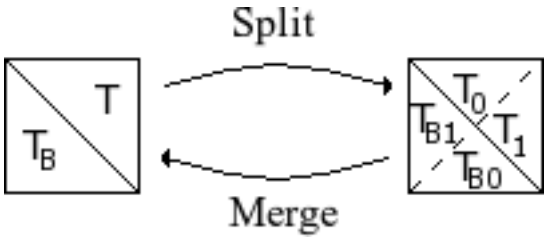

Figura 38: Split i merge sobre una triangulació

un triangle de nivell superior (l més petita) (per exemple el triangle B de la figura 37) no poden ser partits directament i si realment volem partir-los hem de fer abans un split del triangle amb el que comparteix la base. La figura 39 és un cas extrem en el que caldria fer 4 splits forçats abans de fer la partició del triangle T.

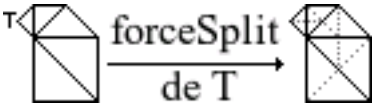

Figura 39: Cas extrem en el que cal aplicar 4 particions forçades

En primera instància es pot pensar que no seria necessari que l'operació de split s'apliquès sobre un diamant i que es podria fer sobre un triangle sol, però això no és possible ja que crearia una discontinuïtat en la superfície. A la figura 40 veiem la representació resultant de partir només un dels triangles del diamant d'una superfície en la que els punts $(0,0)$, $(0,1)$, $(1,0)$ i $(1,1)$ tenen una alçada de 0 mentre que el punt $(0.5,0.5)$ té una alçada de 1.

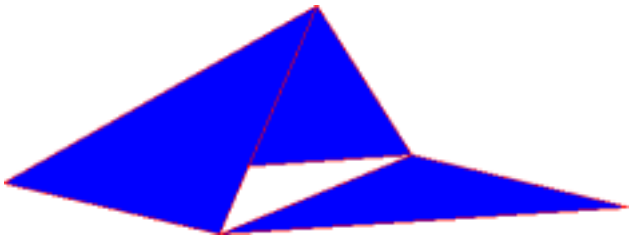

Figura 40: Discontinuïtat: Resultat de partir un triangle i no un diamant

12.5 GPL

GPL (**G**eneral **P**ublic **L**icense) és una llicència de software lliure. La llicència GPL garanteix als destinataris d'un programa informàtic els següents drets o llibertats:

- La llibertat per executar el programa sigui quin sigui el propòsit
- La llibertat d'estudiar com funciona el programa i de modificar-lo (Evidentment cal tenir accés al codi com a precondició per complir això)
- La llibertat de distribuir copies (ja siguin modificades o sense modificar)

La llicència GPL és una de les llicències de software lliure més esteses, si visitem la pàgina de SourceForge (una pàgina web on es registren projectes de software lliure per tal de desenvolupar-los usant els serveis que aquesta proporciona) que llista el número de projectes registrats per cada tipus de llicència, veiem que un 69% (43089 dels 62291) utilitzen la llicència GPL.

El text complet de la llicència es troba a *http://www.gnu.org/licenses/gpl.html*

12.6 Qt

Qt és un *framework* complet de desenvolupament d'aplicacions en C++. Inclou una llibreria així com eines per la internacionalització i el desenvolupament multiplataforma d'aplicacions.

Qt aconsegueix ser multiplataforma abstraient els sistemes operatius i de gestió de finestres, de forma que proporciona una API coherent, lògica i orientada a objectes comú. Les aplicacions Qt s'executen nativament, compilades des del mateix codi font, a totes les plataformes suportades:

- Qt/Windows (Microsoft Windows XP, 2000, NT 4, Me/98/95)
- Qt/X11 (Linux, Solaris, HP-UX, IRIX, AIX, i altres variants Unix)
- Qt/Mac (Mac OS X)
- Qt/Embedded (Linux embedded)

La llibreria inclou una gran part de classes per a la construcció d'interfícies gràfiques, encara que també proporciona moltes altres funcionalitats, com comunicació per xarxa via sockets, classes per treballar amb XML, classes per accedir a fitxers, etc.

Qt té un sistema de llicències dual que permet desenvolupar tant aplicacions lliures (sota la llicència GPL) com aplicacions propietàries. Per poder desenvolupar aplicacions propietàries s'ha de pagar una llicència de Qt per cada programador. Actualment, aquest sistema està disponible per totes les plataformes excepte per a Qt/Windows on només hi ha la possibilitat de pagar per una llicència, però la versió Qt 4.0, esperada per finals del segon quadrimestre del 2005, també usarà el sistema de llicències dual per Qt/Windows.